NAVAL POSTGRADUATE SCHOOL
Monterey, California

AD-A268 937

DTIC
ELECTE
SEP 0 8 1993
S A D

THESIS

The Instrumentation of the Multibackend Database System

by

Andrew Perry Meeks

June 1993

Thesis Advisor: David K Hsiao

Approved for public release; distribution is unlimited.

93-20720

109 pgs

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Computer Science Dept. Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable): CS	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
The Instrumentation of the Multibackend Database System (U)

12. PERSONAL AUTHOR(S)
Meeks, Andrew Perry

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM 06/92 TO 06/93	1993, June 10	109

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Parallel Database, Multilingual and Multimodel Database System, MultiBackend Database Computer, Heterogeneous Database System

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Most database system designs and implementations are limited to single language (monolingual) and single model (monomodel) database systems. No one database language and model can meet every data processing need. As a result, diverse (heterogeneous) database management systems are used in a large organization. This approach results in more processing cost and less data sharing. Therefore, there is a growing need for a solution to the processing cost and data sharing problems of heterogeneous database systems. One solution is a multimodel and multilingual database system (MM & MLDS) as implemented on the Multibackend Database Supercomputer (MDBS). This solution can support an appropriate language and model corresponding to each set of application requirements. In addition, this solution lends itself to data sharing by not only storing all the data in one kernel data model form, but also by allowing cross-model data accessing.

The goal of this thesis is to make the MDBS viable as a kernel database management system. First, we introduce the multimodel, multilingual, multibackend database supercomputer which has been found as a solution towards heterogeneous database sharing. Second, we increase the utilization of MDBS by introducing the system generation software management tool. Finally, we ease the learning curve associated with MDBS by documenting the entire system structure including all the files needed to compile and run the system.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: [X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: David K. Hsiao	22b. TELEPHONE (Include Area Code): (408) 656-2253; 22c. OFFICE SYMBOL: CS/Hq

Approved for public release; distribution is unlimited

The Instrumentation of the MultiBackend Database System

by

Andrew Perry Meeks
Lieutenant, United States Navy
Bachelor of Science (Computer Science), Purdue University, 1985

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1993

Author: Andrew Perry Meeks

Andrew Perry Meeks

Approved By: David K. Hsiao

Dr. David K. Hsiao, Thesis Advisor

Dr. C. Thomas Wu, Second Reader

Gary Hughes, Chairman,
Department of Computer Science

ABSTRACT

Most database system designs and implementations are limited to single language (monolingual) and single model (monomodel) database systems. No one database language and model can meet every data processing need. As a result, diverse (heterogeneous) database management systems are used in a large organization. This approach results in more processing cost and less data sharing. Therefore, there is a growing need for a solution to the processing cost and data sharing problems of heterogeneous database systems. One solution is a multimodel and multilingual database system (MM & MLDS) as implemented on the Multibackend Database Supercomputer (MDBS). This solution can support an appropriate language and model corresponding to each set of application requirements. In addition, this solution lends itself to data sharing by not only storing all the data in one kernel data model form, but also by allowing cross-model data accessing.

The goal of this thesis is to make the MDBS viable as a kernel database management system. First, we introduce the multimodel, multilingual, multibackend database supercomputer which has been found as a solution towards heterogeneous database sharing. Second, we increase the utilization of MDBS by introducing the system generation software management tool. Finally, we ease the learning curve associated with MDBS by documenting the entire system structure including all the files needed to compile and run the system.

Accesion For
NTIS CRA&I
DTIC TAB
Unannounced
Justification
By
Distribution /
Availability Codes
Dist | Avail and / or Special
A-1

DTIC QUALITY INSPECTED 1

TABLE OF CONTENTS

I. THE INTRODUCTION 1

A. MY MOTIVATIONS FOR THE THESIS TOPIC 1
B. THE SCOPE AND OBJECTIVES OF THE THESIS 4
C. ORGANIZATION OF THE THESIS 4

II. A DESCRIPTION OF THE MDBS 6

A. THE ARCHITECTURE 6
B. ITS HARDWARE CONFIGURATION 10
1. Network 11
2. Controller 12
3. Backends 12
C. ITS SOFTWARE CONFIGURATION 13
1. Controller Processes 13
2. Backend Processes 17
D. SUMMARY 17

III. THE ORGANIZATION OF A SYSTEM GENERATION SOFTWARE 18

A. BACKGROUND 18
B. OVERALL DESIGN REQUIREMENTS 19
C. FUNCTIONS OF A SYSTEM GENERATION MANAGEMENT TOOL 19
D. FUNCTIONS IMPLEMENTED FOR MDBS 24
E. THE ORGANIZATION OF MY SYSTEM CODE 24
F. A SUMMARY 25

IV. USING THE SYSTEM GENERATION SOFTWARE 26

A. TO START MDBS 26
B. TO RECONFIGURE THE BACKENDS 26
C. FOR CONFIGURATION MANAGEMENT 26
D. FOR VERSION CONTROL 27

V. AREAS FOR FURTHER STUDY WITH THE MDBS 28

A. UPGRADING EQUIPMENT 28
B. INTEGRATING THE TWO MAIN VERSIONS 29
C. IMPROVING EXISTING MODULES 29
D. THE USER INTERFACE 30

E. IMPLEMENTATION OF MORE CROSS-MODEL ACCESS 31
F. REDISTRIBUTION OF DATA IN AN EXISTING DATABASE 31
G. UPGRADING EXISTING CODE FROM C TO C++ 32
H. MULTIPLE USER AND MULTIPLE DATABASE 32
VI. PROBLEMS 33
A. RETRIEVE - COMMON 33
B. SOFTWARE RELIABILITY 34
VII. CONCLUSIONS 36
A. WHAT I HAVE ACCOMPLISHED 36
B. SOME FUTURE AREAS OF RESEARCH RELATED TO THIS WORK 37
APPENDIX A. FILE ORGANIZATION OF MDBS 39
A. CONTROLLER 40
B. BACK END 46
APPENDIX B. HOW TO START AND RECONFIGURE MDBS 48
A. TO START MDBS 48
B. TO RECONFIGURE MDBS 52
C. TO DISTRIBUTE BACKEND EXECUTABLE CODE 53
D. TO CHANGE VERSIONS 55
E. TO LOAD MDBS ON A NEW MACHINE 57
F. SAMPLE ERROR MESSAGES 58
1. Invalid Option 58
2. Incorrect new version name 58
3. Incorrect version for pwd 58
4. Include file out of date 59
5. Missing file configuration file 59
6. Missing or Incorrect process names 59
APPENDIX C. ON-LINE DOCUMENTATION 60
APPENDIX D. FLOW CHARTS FOR SYSTEM GENERATION SOFTWARE 62
APPENDIX E. SOURCE CODE FOR SYSTEM GENERATION SOFTWARE 67
APPENDIX F. MISCELLANEOUS FACTS ABOUT USING MDBS 86

A. HOW TO START-UP THE SYSTEM AFTER THE POWER GOES OFF. 86
1. For db1 - db9 86
2. For Sun4 Workstations 86
B. SOURCE DIRECTORY LOCATION 87
C. COMPILING 87
D. COMPILING AN INDIVIDUAL PROCESS 89
E. DEBUGGING AND FLAGS 89
F. THE HIGHER LEVEL LANGUAGE INTERFACES 90
G. CHECKING THE HARDWARE CONFIGURATION 91
H. THE DEVELOPMENT CYCLE 92
I. RUNNING THE SYSTEM 92
J. UTILITY PROGRAMS 93
APPENDIX G. SELECTED MDBS FILE LISTINGS 94
LIST OF REFERENCES 99
INITIAL DISTRIBUTION LIST 100

LIST OF FIGURES

Figure 1, The Multimodel, Multilingual and Cross-Model Accessing Capability.....7
Figure 2, Architecture of the DBC.....9
Figure 3, The Multibackend Database Supercomputer (MDBS).....11
Figure 4, Interrelation of System Processes.....14
Figure 5, The Multimodel and Multilingual Database System.....15
Figure 6, The Model-Language Interfaces on MDBS.....16
Figure 7, Directory Structure of the MDBS files on the controller.....39
Figure 8, Directory Structure on Backend Machines.....46
Figure 9, Documentation of MDBS design and structures.....60

ACKNOWLEDGEMENTS

I would like to thank Dr. David Hsiao for his time and patience while I learned the Multimodel and Multilingual Database System on the Multibackend Database Supercomputer (MDBS) and later while he reviewed the thesis. Without his knowledge and expertise, this project and thesis would not be possible.

I would also like to thank the support staff at the Naval Postgraduate School for their help. Particularly John Locke and Susan Whalen for their assistance with the MDBS.

I. THE INTRODUCTION

A. MY MOTIVATIONS FOR THE THESIS TOPIC

Over the past few decades there has been a rapid increase in storage[1] and processing[2] capacity of computers (Elmasri, 1989, p. 637). While the capacity increases, the cost for the same amount of capacity decreases. One would expect the overall performance for computer programs and systems to improve. This has been the case for the general-purpose computer; however, one application for the general-purpose computer has not kept up with the growth. This application is called database management. In the following references we note that database systems have unique hardware needs that are not normally built into general-purpose computers.

> ...the overall performance of database systems is still a great concern. [There is a]...mismatch between access speeds of secondary storage devices and those of central processing units leading to inevitable delays in processing large volumes of data stored on disk. Compared to the advances in speeds of processors...the rate of increase of speeds of secondary storage devices (mainly disks) have been slow. (Elmasri, 1989, p. 637-8)

> Typical database management operations require the processing of 90-95 percent of related data for the purpose of producing 5-10 percent of useful information.... (Banerjee, 1979, p. 137)

> ...data is not stored at the place where the data is processed. To 'stage' the data in the main memory for processing is very time consuming. It often ties up important resources of a computing system such as communication lines, channels, and data buses. Ideally, data should be processed at the place where they are stored to avoid spending time in moving data between the main memory and the secondary memories. (Su, 1979, p. 430)

Methods to improve the performance of database systems can be considered in two main categories: software and hardware. The software-only approach uses indices, file-

---

1. Over the past 20 years, data storage capacity for a moving head disk has doubled every three years and Dynamic Random Access Memory storage density has quadrupled every 3 years (Hennessy, 1990, p. 17).
2. Growth of Performance over the last 20 years ranged from 18% to 35% depending upon computer class (Hennessy, 1990, p. 3).

organization techniques, intra-data pointers, data compaction, directories, cross-reference pointers, computed addressing, to name a few, which alleviate the speed problem to a certain extent (Su, 1979, p. 430). Unfortunately the software-only approach brings with it additional cost. The cost and overhead are the requirement for more storage and increased processing time. The overhead includes, for example, maintenance of indices, computation of pointers, and performance of compaction algorithms.

A second approach is the hardware approach. Several database machines have been proposed over the last two decades. The following is a list of some of the proposed hardware concepts: *associative memory* (DeWitt, 1979), *backend* computers, *cellular logic* (Schuster, 1979), *cross-bar network* (DeWitt, 1979), *function-specialized* (Banerjee, 1979), *logic-per-track* (first introduced by D. L. Slotnick), (Su, 1979), (DeWitt, 1979), (Schuster, 1979), *multiprocessor* systems (DeWitt, 1979), and *special purpose processors* (Su, 1979), (Banerjee, 1979). Few have been successfully built and even fewer have become commercial products.

The ideal database computer should use the optimal software techniques combined with the optimal hardware designs. The ideal database computer has not been built. One database machine, known as the Multibackend Database Supercomputer (MDBS), is at the Naval Postgraduate School's Laboratory for Database Systems Research. This system which started development in the early 1980's, is fundamentally based upon the Database Computer (DBC) described in (Banerjee, 1979). This system not only addresses a solution to the problem of processing large amounts of data efficiently, but is also a multilingual and multimodel database system.

What is the significance of a multilingual and multimodel database system? Consider large organizations which often use multiple database models and languages to meet their database needs. Three typical databases would be for inventory, personnel, and product assembly. An inventory (consisting of suppliers, supplies, and orders) can best be modeled with a network database system. A personnel database can best be maintained with a relational database system. And a product assembly (consisting of products, assemblies,

sub-assemblies, parts, and so on) can best be supported by a hierarchial database system. In addition to the different models and languages, there are often more than one database with the same information. One example is a personal database for a social roster on one machine and a payroll database on another machine. Both databases have a person's name and address among other information.

Some results of having different models and languages are as follows. *First* there is a lack of data sharing. Since the heterogeneous databases[3] cannot communicate between themselves, all information must be maintained by each system even if the same data exists in another database system. The lack of data sharing increases the storage capacity needed to maintain the separate databases. *Second*, the duplicate information stored leads to data inconsistency problems. If one updates a database, then another database with the same information will be incorrect until it is also updated. *Third*, personnel who know how to use one database model and language have to be retrained to use another model effectively. This specialization makes it more costly (and harder) to move personnel between jobs, since it requires knowledge of a database model with which the personnel are not familiar. This problem affects the total cost and effectiveness of maintenance and support of database systems.(Hsiao, 1992, pp. 127-132), (Chung, 1990, p. 70)

The fact that different models exist does not mean that we should look for a single or universal data model and language so that we can convert all present and future databases into one model and language just to cut the cost of maintenance and support. At the same time we should not ignore the problem. One solution to the heterogeneous data sharing problem was proposed by (Hsiao, 1989). This solution has been partially implemented on the Multibackend Database Supercomputer (MDBS) at the Naval Postgraduate School (NPS).

Recognizing that no one database model or language can meet all database needs, this solution supports all database languages. All data is stored using a kernel data language (the

[3]. Heterogeneous in the sense that there are multiple data models.This is not to be confused with different dialects of the same model and language. Nor does refer to different hardware.

Attribute Based Data Language) and requires that each database language interface store its data using the kernel data language. This design was chosen since it reduces the cross-model data translations from n^2 - n to n - 1. Where n is the number of database languages implemented by the system. (Hsiao, 1992)

B. THE SCOPE AND OBJECTIVES OF THE THESIS

My objective of the thesis is three fold. First, to introduce the multimodel, multilingual, multibackend database supercomputer which has been found as a solution towards heterogeneous database sharing. Second, to introduce the system generation software management tool. And finally to document the structure of the MDBS files needed to compile and run the system.

There should be an automatic way to generate, manage, and control various versions of a database system with little or no human intervention and support. This would be good for the systems administrator or developer, but is of little concern to the user. Since there are few parallel and distributed database systems in existence, there is a lack of methodology in developing these systems. On the other hand, most contemporary database computers are not parallel; thus they do not have multiple backends. There is little need to distribute multiple copies of the same code to multiple backends on contemporary database computers, since they do not have backends. Methods for version control and configuration management of system software for the scalable and parallel database computer must be developed to simplify the design process and final usability of the system. Thus methods of design and implementation of a system generation management tool for a specific parallel and multibackend system must be developed. In this thesis, I will demonstrate the feasibility of constructing such a tool on our research database computer, MDBS.

C. ORGANIZATION OF THE THESIS

In Chapter II of this thesis, I provide an overview of the hardware and software of the Multilingual, Multimodel, Multibackend Database Supercomputer (MDBS). In Chapter

III, I describe what system generation software is and what functions it should provide. In Chapter IV, I describe how my system generation software is implemented and how the code is organized. Areas for future study are addressed in Chapter V. The appendices contain much information about the MDBS software which has not been put into one place before.

II. A DESCRIPTION OF THE MDBS

A. THE ARCHITECTURE

There are two aspects of the MDBS which make it a powerful database computer. The first aspect is the multi-lingual/multi-model capability and the second aspect is scalability with parallel backends.

As described in the previous chapter, MDBS uses the Attribute Based Data Language (ABDL) as the kernel or common data language to store all databases. Figure 1 shows the relationship between various database users and how a database is viewed and stored. There are five[1] language interfaces on the system, attribute based, network, hierarchial, relational, and object-oriented. Each the language interface uses a schema to view a particular database (which it can access). Limited cross-model[2] capability exists on MDBS, relational to hierarchial and relational to object-oriented.

The parallel processing capabilities of MDBS were based upon the Database Computer (DBC) described in (Banerjee, 1979) and roughly depicted in Figure 2. The Database Computer had many optimizations built-in to make it as quick as possible in processing database queries. Some optimizations which DBC used were (1) modified moving head disk controllers which could data from all each tracks in one cylinder simultaneously, (2) partitioning to evenly distribute the base data to ensure maximum number of heads (or tracks) in each cylinder are being used, (3) arranging then processing queries in conjunctive normal form, and (4) clustering of data. The DBC architecture broke query processing into several stages and placed single processors at four stages and multiple processors at the base data and where the indices are stored and processed. The four units with single processors are the database command & control processor (DBCCP), keyword transformation unit (KXU), index transformation unit (IXU), and security filter

1. A sixth one, functional, is in the process of being added.
2. A database user can access (retrieve, update, delete, or insert) one database using a different language than was used to create and/or update the database. For example, a hierarchial database can be accessed by the relational language interface through a one time schema transformation.

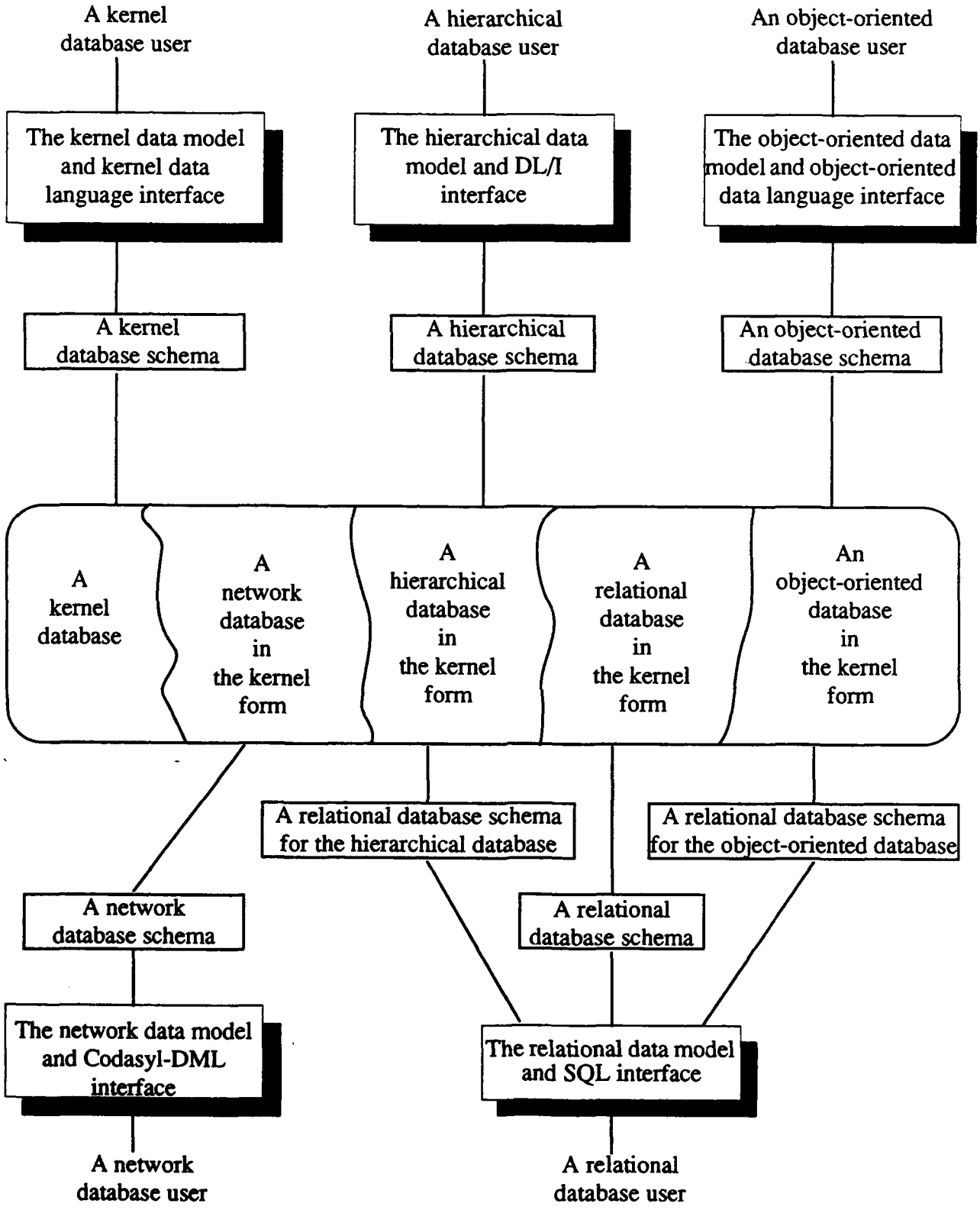


Figure 1. The Multimodel, Multilingual and Cross-Model Accessing Capability

processor (SFP). Multiple processors were placed the main memory (MM), the structure memory (SM) and the structure memory information processor (SMIP). The MM is a moving head disk drive with a controller modified to simultaneously read and process data from all tracks in one cylinder. The SM can be either bubble memory or charged coupled devices (CCD's). There is one processor per track in the SM and one processor per memory unit in the SMIP. These processors worked in parallel on various stages of query processing. The structure loop was designed to limit main memory search space, determine authorized records for accesses, and for clustering records received for insertion into the database. Two advantages of interest of the DBC are (1) an estimated speed up of over 80 times that of a database system implemented on a general purpose computer (single CPU, single disk machine) and (2) the method of storing data in partitions was intended to segregate data as it was stored for a multi-level security database[3].

For various reasons, the DBC was never built. In 1980 Dr. Hsiao decided to use "off the shelf" equipment to implement a simplified version of DBC. This version only used one processor (per controller and per backend) to perform the tasks that had been designed to be performed by the multiple processors of the DBC design. Since the disk controllers were not modified, the data could not be processed as it was read from the disk. Without the multiple processors on each backend there was a significant loss in the speed-up that would have occurred with the DBC design. This system was modified and expanded and then moved to the Naval Postgraduate School in 1983. It is now known as the Multibackend Database Supercomputer (MDBS).

One strength of MDBS is the method used to store database data on the backends. As a database is loaded, it is divided into clusters of similar data descriptors[4]. Using clusters,

---

[3]. The multi-level security topic is not related to this thesis, but is very relevant to a Department of Defense (DOD) application. The access authorization for a user or process would determine which partitions of data were within the authorized clearance level for that user or process. A good design for storing classified data on the DBC would not allow reading any unauthorized data from the data disk.

[4]. A descriptor is an attribute value or range of values and is used to group the data.

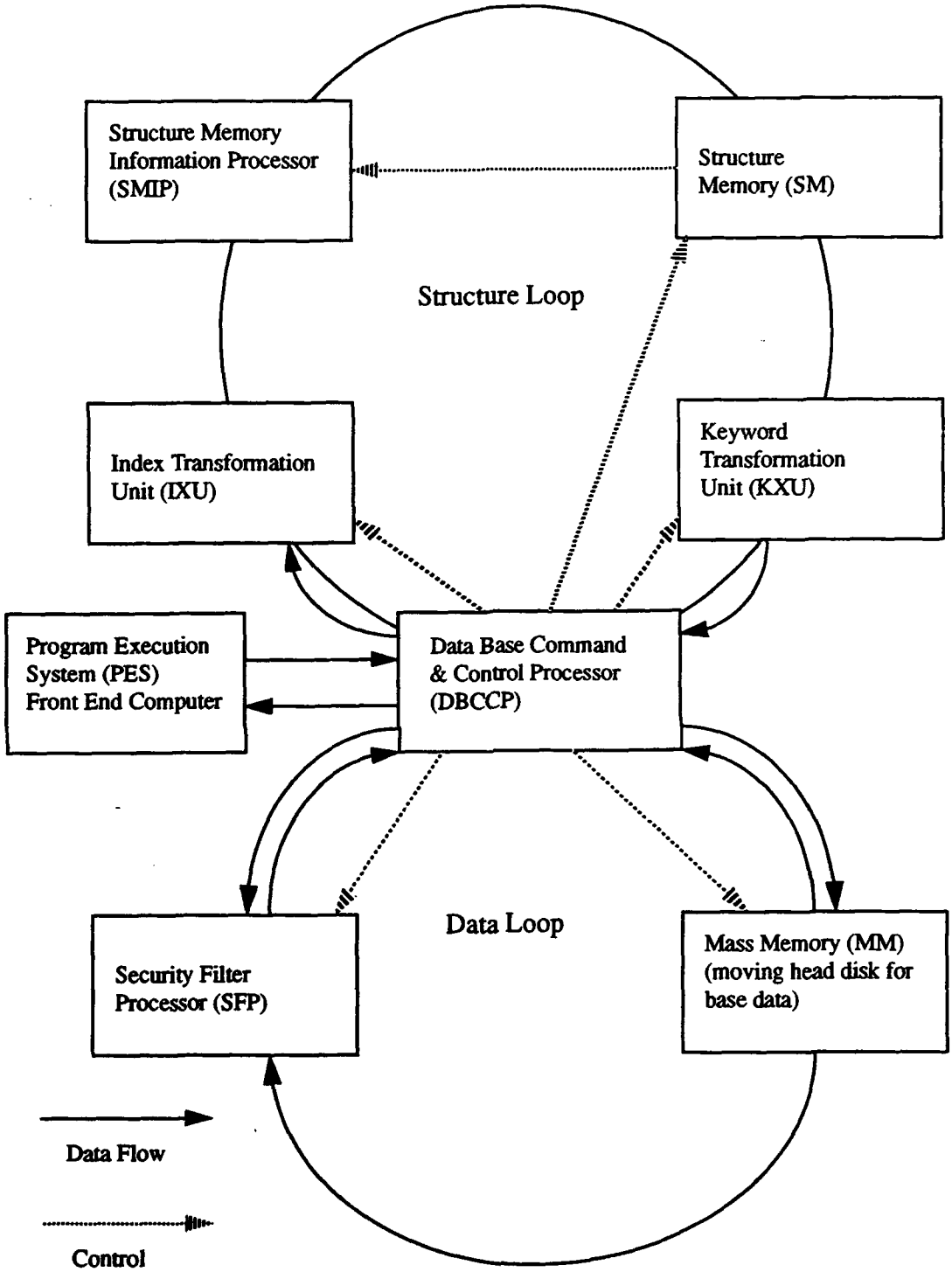


Figure 2. Architecture of the DBC

the base data[5] is spread evenly across the backends. This even division of data across the backends maximizes parallel query processing on the backends.

A second strength of MDBS lies in its ability to easily change the database storage capacity and the amount of parallelism. This can be done by changing the number of backends in the configuration. As a database grows over time, reaching the capacity of an existing system, the *traditional solution* is to get a bigger and faster computer. Once the replacement system is found then (1) a database system must be installed, (2) the data must be reloaded on the newer system, and (3) the existing queries must be rewritten and tested for the new system. Solving the database growth problem on the MDBS is much simpler. To enlarge the MDBS, one buys more backend machines, backs up the database, adds the new backends to the configuration, and reloads the database on the new configuration. By reloading the database, its data is now redistributed over all backends for the new configuration and is ready for use.

With the database distributed across the backends, each time the controller sends a query to the backends, each backend does some work in parallel. Thus the parallel processing capabilities are achieved on MDBS by the multiple backend capability and to a lesser extent by having separate disks for base data, meta-data[6], and paging.

B. ITS HARDWARE CONFIGURATION

A fairly detailed description of the multibackend database supercomputer (MDBS) can be found in (Hsiao, 1991). Figure 3 shows an overview of the configuration. At the time of this writing, the MDBS is being ported to a new controller[7] and backend[8].

---

5. Base data is stored as attribute-value pairs using the Attribute Based Data Model (ABDM).

6. Metadata is information about the base data. To access the base data to find the appropriate cluster(s). The backend can then directly access the base data using the cluster identifier(s). For more details see (Hsiao 1991).

7. The new controller is a Sun model 4/110 workstation called db11 which has the Sparc 4 RISC architecture with 8 Megabytes of random access memory (RAM) and one 373 Megabyte hard drive.

8. The two backends, db12 and db13, are Sun model 4/280 workstations which each have Sparc 4 RISC architecture with 16 Megabytes of RAM, one 96 Megabyte disk for paging, one 96 MB disk for meta-data, and one 1000 MB hard disk for data.

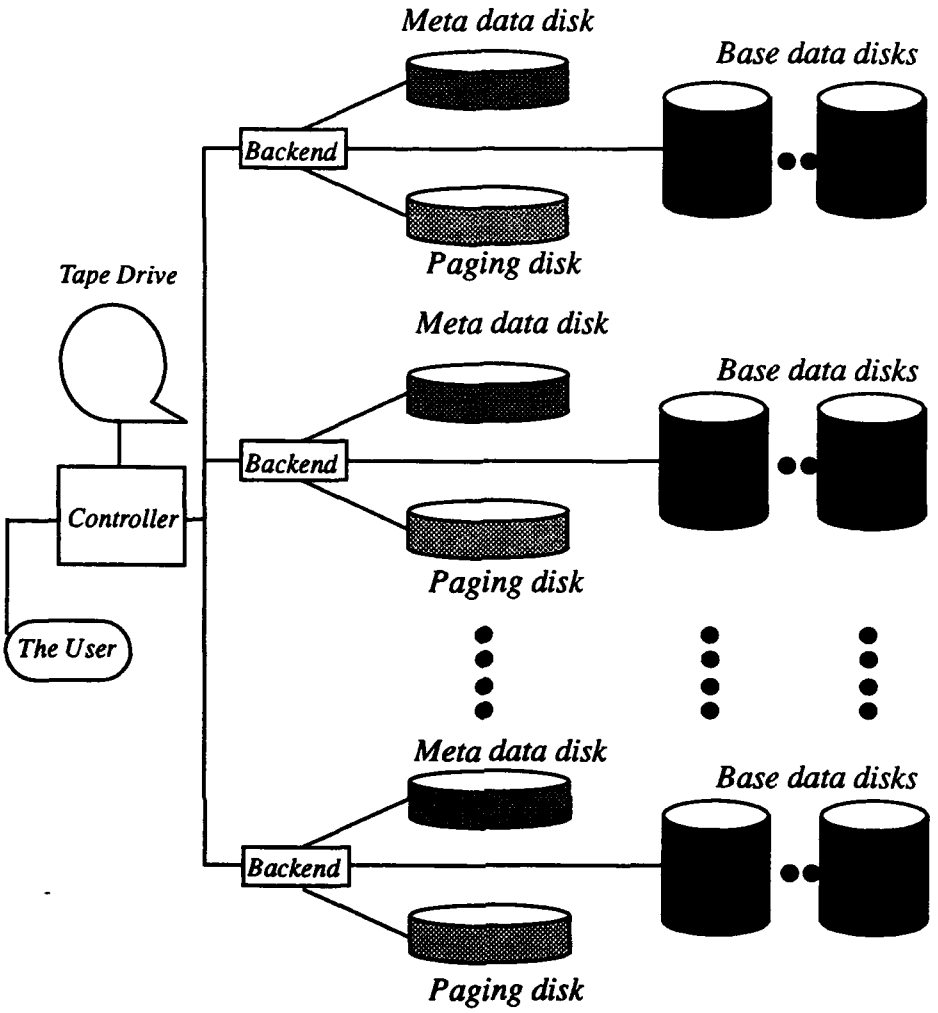


Figure 3. The Multibackend Database Supercomputer (MDBS)

1. Network

The local area network (LAN) consists of the Ethernet cable, the backplanes and their transceivers. There is one LAN for the MDBS. The controller, db8, acts as the gateway to the local computer science LAN. Each backplane has an Intel 80186 microprocessor and 128-Kilobytes of real memory. The 80186 processors work in conjunction with another processor, an Intel 82586, which supports the transmission control protocol/internet protocol (TCP/IP). The Ethernet cable has a transmission rate of

10 Mega bps. The Ethernet allows messages to collide, but the user defined protocol (UDP) software helps make unreliable broadcasting reliable.

2. Controller

The controller supports the user interface as well as communications with each backend. The controller is an Integrated Solutions (ISIV) model V24S workstation with a 16 MHz 68020 microprocessor, 4MB ram, a 96 Megabyte paging disk to support virtual memory, 96 Megabyte disk for system and user files, and a tape drive. The controller is connected to the backends via one backplane and to the local computer science department network via a second backplane. When the database system is executing, the controller communicates with the backends via sockets. The magnetic tape drive on the controller can be used for mass loading of a database, back-ups of a database, and normal back-ups of all existing files.

3. Backends

Although MDBS is capable of supporting many backends, there are three backends[9] in the network. Each backend is an Integrated Solutions (ISIV) model V24S workstation with a 16 MHz Motorola 68020 microprocessor, and three disk drives. One 96 Megabyte Winchester-type disk drive is used for paging, another 96 Megabyte disk is used to store the meta-data. The base data is stored on a 500 Megabyte moving-head disk. Communications with the controller are handled by the backplane which communicates with the controller and other backends via the Ethernet. Each backend maintains the meta-data table for the entire database. The backends for the database computer are not required to be identical machines.

---

[9] The system had as many as eight backends. With two backends on loan and three suffering from old age, there are only three which can be used.

C. ITS SOFTWARE CONFIGURATION

The MDBS uses UNIX[10] Berkeley Software Distribution (BSD) 4.3 operating system. All of the software is stored and compiled on the controller. When the backend code is compiled it must be distributed from the controller to each backend. To make the compilation and distribution process simpler, identical CPU and operating systems are used for contoller-backend combinations under development. On-line backups of code can be stored on the backends. To support MDBS, there are six processes which run on the controller[11] and six other processes which run on each backend[12] in parallel.

1. Controller Processes

The six processes on the controller are the user interface (TI[13]), controller get (CGET), controller put (CPUT), request preparation (REQP), post processor (PP), and insert information generator (IIG). A simplified diagram of how the six processes interrelate is in Figure 4.

The process--by the far the largest--is TI which implements the data language interfaces described at the beginning of this chapter and the cross model capabilities of the MDBS. This process maps each data model and language to the kernel data language (KDL) and kernel data model (KDM). On MDBS, kernel data is stored using the attribute based data model (ABDM)[14] and attribute data language (ABDL). For each data language, there are four software modules which make up the user interface (illustrated in the shaded

---

10. UNIX is a registered trademark of AT&T Bell Laboratories, Inc.

11. Appendix A contains a directory tree of all the files in the MDBS. Specifically, the controller processes (cget, cput, iig, pp, reqp, ti) are located in the directory Version_Name/CNTRL/ and are compiled from source code in the corresponding subdirectories (CCOM, CCOM, IIG, PP, RECP, TI) of CNTRL/.

12. Appendix A contains a directory tree of all the files in the MDBS. Specifically, the backend processes (bget, bput, cc, dio, dirman, recp) are located in the directory Version_Name/BE/ and are compiled from source code in the corresponding subdirectories (BCOM, BCOM, CC, DIO, DM, RECP) of BE/.

13. TI stands for test interface, which has become the normal user interface.

14. The ABDM was chosen for various reasons including separate modeling of base data and meta data, the ease of clustering base data into mutually exclusive sets for storage on backends. Its semantic richness allows for other languages to be translated into the ABDL.

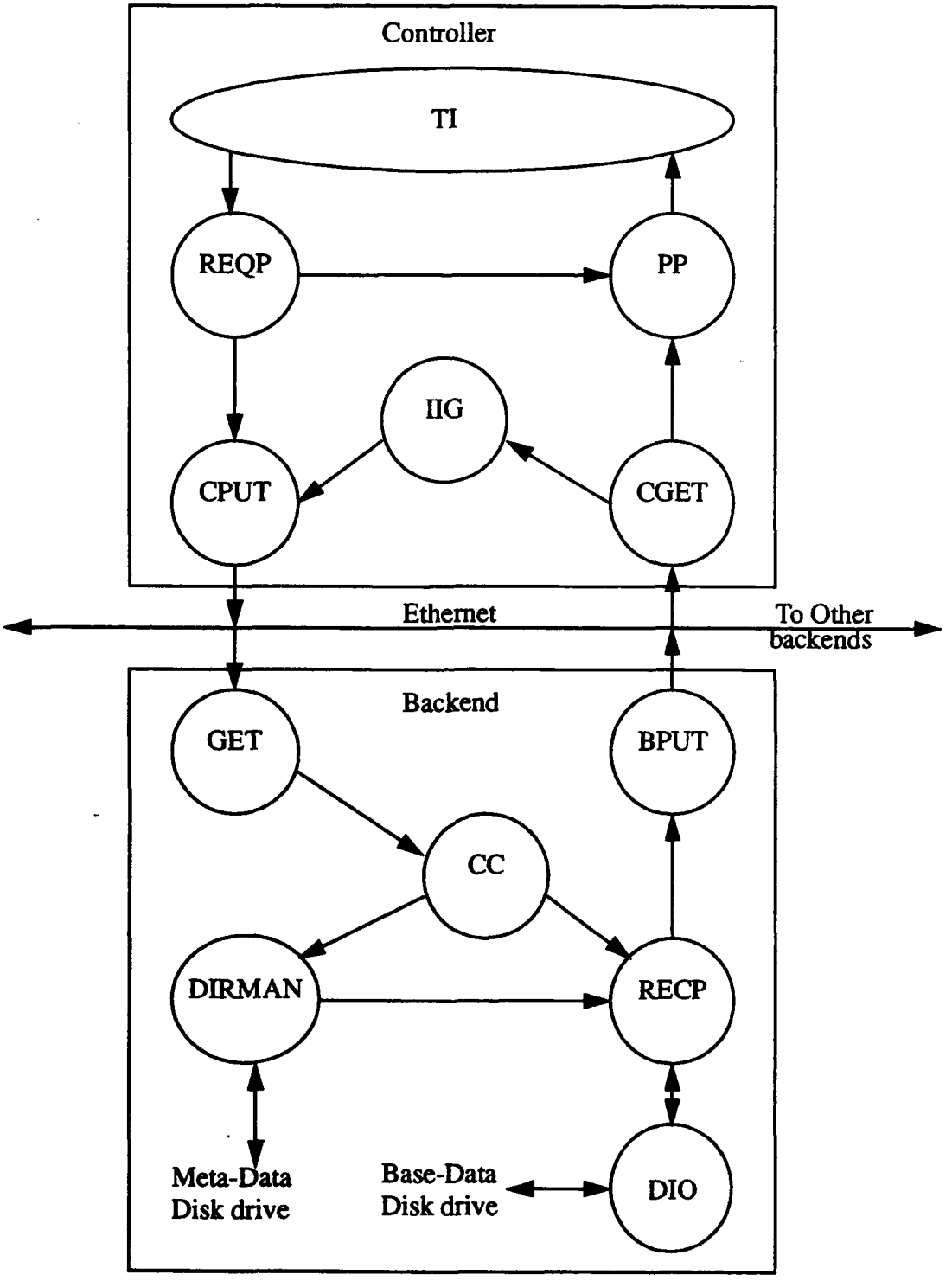
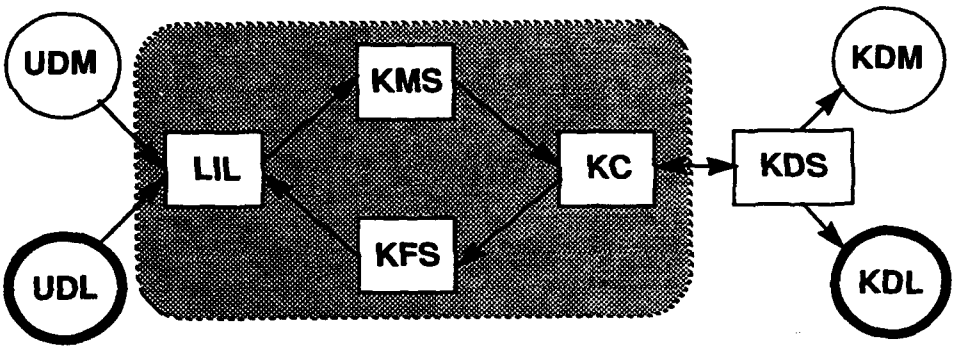


Figure 4: Interrelation of System Processes

area of Figure 5). These are the language interface layer (LIL), the kernel mapping system



UDM : User Data Model
UDL : User Data Language
LIL : Language Interface Layer
KMS : Kernel Mapping System
KFS : Kernel Formatting System
KC : Kernel Controller
M/LI : Model/Language Interface
KDS : Kernel Database System
KDM : Kernel Data Model
KDL : Kernel Data Language

Figure 5. The Multimodel and Multilingual Database System

(KMS), the kernel formatting system (KFS), and the kernel controller (KC). When starting MDBS, the user chooses which data model and data language interface with which he wants to view a database. The model he chooses is called the user data model (UDM) and the language corresponding to that models the user data language (UDL). Once a language has been chosen, the user communicates with the LIL corresponding to that language (see Figure 6). For a detailed description of how the software modules interact, see (Bourgeois, 1993, pp. 7-24).

Communications with the backends are handled by two processes, CGET and CPUT. The CPUT process sends database creation and queries to the backends. The CGET process receives acknowledgments for creations and receives the results of the queries from

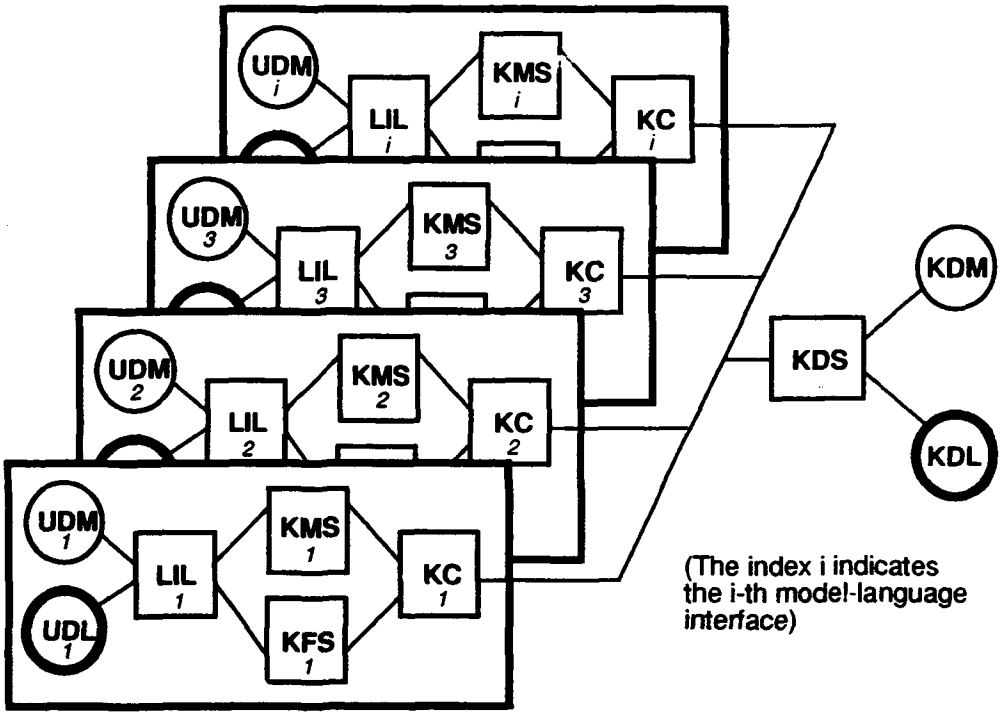


Figure 6. The Model-Language Interfaces on MDBS

each backend. The REQP process receives database transactions from the KC in the TI process and sends them to the backends through the CPUT process. The PP coordinates responses from the backends and returns the results of a user query to the KC in the TI process.

The IIG process is used to determine which backend will insert a particular record. During an insert operation, the IIG process tells a specific backend--via the CPUT process--whether to start a new track or put the record being inserted into an existing track. The IIG process uses a space utilization table from which it can determine for each cluster which backend contains the last full track of the cluster, and can determine the first available track for inserting a new track of clustered records.

2. Backend Processes

The six processes on each backend are backend get (BGET), backend put (BPUT), concurrency control (CC), data disk input/output (DIO), directory management (DIRMAN), and record processor (RECP). Each oackend communicates with the front end and the other backends via BGET and BPUT. The CC process allows concurrent database access by multiple requests. The DIO process deals with disk addresses of data and optimizes disk accesses. The DIRMAN process accesses the meta-disk to perform look-ups in the descriptor and cluster tables to produce disk addresses. Finally, the RECP process performs data retrieval, storage and processing required on whole records and sends its output to the BPUT process.

D. SUMMARY

There a multitude of computer systems in industry, government, and education today. With the growing expense of maintaining these systems, there is a greater interest in communication between heterogeneous databases. The Multibackend Database Supercomputer, design offers one solution by storing all databases using one language, a kernel data language, and providing multiple model/language interfaces to the users of the databases. An added benefit of MDBS is its scalability which is inherent to its parallel backend design.

III. THE ORGANIZATION OF A SYSTEM GENERATION SOFTWARE

A. BACKGROUND

Computers are being used to perform many tasks which we used to do. The examples are endless, from word processing to database management, from computer aided design in manufacturing to process control, and from monitoring someone on a life-support system to monitoring the skies for incoming missiles or "hostile" airplanes. As computers continue to become more powerful, they can be used to perform even more impressive tasks. It is important to note that when you compare a computer to a human, the computer will be superior in several categories. Compared to human beings, computers are much cheaper, they can perform various functions quicker and more accurately, with well written programs they are more reliable, they are free from boredom, they can analyze input quicker, they can produce output quicker, they do not need to unlearn a previous task or do they require training for a new task, they can operate in hostile environments (hot, cold, high radiation, no atmosphere), and they do not have personal difficulties (family, marital, vacations, pay & benefits, or deaths to deal with).[1]

The thesis is not about to suggest that we should or even could completely replace humans with computers. Instead we should make computers do as many tasks currently done by humans as we can. One focus of the System Generation Software is automating the tasks that can be performed by a computer. The benefit of automation is to give more time to the operator or user so they can do their primary task of testing the system or using a database.

A project administrator is responsible for dividing up the work, assigning it to individuals, monitoring progress of the work, and then ensuring the pieces are integrated into a working product. An administrator should not have to manually create working

---

[1] Lecture notes by Richard Hamming for the course EC4000, "Future of Technology," at the Naval Postgraduate School, Monterey, CA, 1993.

copies of code for program developers to use. Nor should an administrator be required to manually keep track of all code needed for a particular version.

B. OVERALL DESIGN REQUIREMENTS

From a user's point of view there are three important requirements for any software. They are ease of use, doing what you expect the program to do, and reliability. From the point of view of a systems programmer or someone who must maintain the code, an added requirement is well written code.

Ease of use is the important requirement for any software. Given a choice, if you have to use a system that is not easy (or intuitive) to use then you will probably avoid that system. If a system is hard to use, then you spend more time learning how to use that system thus less time getting your job done. This is true for any job whether it be supervising, data entry, data queries, software maintenance, or software improvement. With a system that is easy to use, the more work the program can take from the supervisor, developer, or user, then the more useful work they can perform.

C. FUNCTIONS OF A SYSTEM GENERATION MANAGEMENT TOOL[2]

Some repetition in our lives is inevitable. Every work day many people wake up, get dressed, eat breakfast, drive their car to work, and so on. However, when we have to repeat things on a computer, we create macros, aliases, or executable files to get the same work done with less human effort. There is no reason to not apply this concept in every computer application. In the world of computers, there are many examples of how a programmer can make using the computer easier. Macros, aliases, and executable files are just a start, graphic interfaces were made possible after the invention of the mouse. The absence of a command line on Apple Computer products saves the user from remembering specific commands and syntax which are a burden to users of UNIX.

---

[2] Back ups were not listed as a function performed by system generation software since a back up process or procedure should already be performed on a routine basis.

There is no reason to not automate as much of the software development and testing process as possible. A system generation system should be able to create and manage multiple versions of software. Managing versions includes the version name, who is working on the version, who is authorized to modify the files, where the code is located, and how to compile the code. I am not going to discuss personnel management or security issues in the thesis so I will limit the discussion to managing the versions. When multiple versions exist on a system which consists of multiple machines, there is the added requirement to coordinate running the same version code on each machine.

With MDBS there are two things to manage. The first one is the controller-backend configuration. The second one is controlling the version of the software being used.

The, controller-backend configuration is primarily a concern during system start-up. Starting MDBS takes about 13 commands (approximately 450 characters) for the controller and about 6 commands (approximately 450 characters) for each backend. As with most computers, there is no allowance for typographical errors. It is unrealistic to expect anyone to type all commands each time he runs the system with backends. The first step to reduce the number of keystrokes needed to start the system is to group commands into shell scripts[3]. However, since there are multiple combinations of backends to use for a particular run, the number of possible shell scripts becomes enormous[4]. If you want to erase an existing database from the data and meta disks in preparation for loading a different database or starting over, you have even more commands to type. There is added work of navigation through directories and machines. To execute a command on each backend you need to rsh or specify the full path name (e.g. /@db3/usr/work/mdbs/bin/zero devs1). Other complications include the fact that one process on each machine cannot be started until the other five processes on that machine are executing properly. And one process on each machine expects to hear from other processes on that machine within a specific time frame

---

[3]. similar to .BAT files on IBM compatible personal computers.

[4]. Using one controller and every combination of 1 to 8 backends, there are 255 possible backend configurations.

or it will terminate. Developing software and queries for the system often results in frequent system start-ups. When you use multiple backends, keeping track of the whole system becomes a hinderance to anyone who concentrates on what he needs to do rather than to get the system running. This suggests automation to start the MDBS to keep track of the backend configuration.

Version control is needed while improving the software of any project. When a portion of the system needs to be added, changed or improved then a decision must be made about what will be modified. There should be one copy of the original code and another copy of code that will be modified. This can be done in several different ways. Three methods are presented for discussion: (1) make a back-up and modify the on-line version, (2) copy the entire code for the system and modify the copy, and (3) copy a minimal subset of the entire code and modify the copy (using as much of the original code as possible).

For discussion purposes, the first two are similar. By making an entire copy of the system, you have just made it difficult to incorporate the change back into the original, unless you plan on abandoning the original version and staying with the new version. Methods (1) and (2) make sense if the system consists of only one segment of code being modified or you are the only person working on the entire system and can keep track of all changes. Now lets say that there is more than one segment of the code being modified. Will you copy the entire system for each version and have multiple revised versions to merge into one version for the final product? This method of providing copies of the entire code for each version causes the amount of code on-line to be directly proportional to the number of versions being modified. Some would argue that memory is cheap so we should not be concerned about multiple copies; however, more important than memory is time. The more copies of code one has to integrate, the longer it takes to integrate[5] as a result, the overall cost is higher.

---

[5] Something we discovered the hard way when attempting to merge the code from version 7 with VerE.6 of the MDBS.

Now consider the third approach. Say the original software is partitioned in some manner such as a user-interface, common code, debugging code, system maintenance, and backend code. If software from one partition is being worked on then only that partition of files needs to be copied for someone (or a group of programmers) to modify. To test the partition of the code under development, one compiles the partition of code being modified using any include files and object code needed from the unmodified partitions. Then, if needed, link the original code with any newly created object code. Once all the executable file(s) affected by the modifications have been compiled, then the system is ready for execution. By copying only the partition of code that is being modified, *first* you will still have the original working version available, *second* you will not use as much disk space as used by the second method, and *third* you will make the job of re-integrating modified code into the original easier since you only have one partition to integrate for each version.

For the sake of discussion, let us say that one partition being modified is for the backend and that no other partition depends upon it. To test the whole system you compile the process just modified. Assuming there are multiple processes in the original system, start the system using the executable code for the one process just compiled and the executable code for the processes from the unmodified code.

Ideally, the System Generation Software can be designed to automatically select only the portions of code needed for development and not replicate unneeded code from an original version to a working version. To do this would require a complete dependency graph of all files in the system and some means of knowing what files to copy for every possible change to the system. This approach could be automated with considerable effort. Two questions arise, "Is it worth the effort?" and "It there an easier way?"

An example from the MDBS system follows. On db3 was a working copy of the multilingual database system[6]. This version did not have an Object-Oriented interface and there were three thesis students ready to implement one. The VerE.6 was protected so no

[6] in directory /usr/work/cs4322/VerE.6

one would modify the code and a new directory[7], was created with the same sub-directories as VerE.6. After determining that none of the backend processes and only one of the six controller processes[8] would be affected, the entire code for that process was recursively copied to the directory, /usr/work/mdbs/rich/CNTRL/TI. The files copied included source code and make files for the Language Interface. To compile the system, the main make file was modified to only compile the files needed for the language interface process. There were eleven processes not affected by the modifications. So to test the system, the shell script to run the system was modified to start the eleven processes from the previous version and the one process compiled from the modified code.

Looking back, there was not need to copy the code from all language interfaces such as relational, hierarchial, and network; however, the more time would have been needed to further modify the make files to support copying only a portion of the language interface. This raises the question which a supervisor should keep in mind, "What is the optimal amount of code to copy for a new version?" If you copy the complete system code, then you have a large amount of code to integrate back into the original system. If you copy the minimal amount then you may end up using more time setting up the files and make files than you could save by not having to integrate the modified code into the original system. A supervisor must exercise "good" judgement in making this decision. As advanced as computers are, they are not ready to make such decisions.

There are ways to assist a supervisor in making these decisions. Case tools and some compiler packages generate dependency graphs automatically for files. The dependency graph for files is not always visible to the user even though the system uses that information to compile the files for a system. A less reliable source are the dependencies which exist in a makefile either constructed manually for C files or automatically for some compilers.

---

7. /usr/work/mdbs/rich,
8. dblti.out,. the language interface.

D. FUNCTIONS IMPLEMENTED FOR MDBS

The four functions performed by my system generation software management tool are start-up, reconfiguration, distribution of code, and version creation. All the commands needed to *start* the MDBS are performed automatically by running the system generation software management tool program. To change the number or names of backends in a system configuration, the system generation software can *reconfigure* which backends will be used for the next start up of MDBS. When modifications are made to one or more backend processes, the recompiled executable code must be copied or *distributed* to each backend. And finally, after someone copies the applicable portions of code and modifies the configuration header file as described in the source code comments (see Appendix E) and modifies the makefiles as discussed earlier in this chapter, one can *create a new version* of the MDBS software on the controller and set up each backend for the execution of processes created by the new version.

E. THE ORGANIZATION OF MY SYSTEM CODE

To automate the start-up and some management of MDBS, I broke up the start up commands and made them generic. Backend commands will work on any backend. On each backend I put commands common to every version for the backend into the bin directory. Since different versions are kept in different directories, I decided to include the version name in the directory name on both the controller and the backend. On the backend, the directory for the executable code must have the name be.<version_name>. For example, be.greg contains the 6 processes and the run.be command which calls them. To automate creation of the shell script which calls the 6 backend processes, I made a template file which is edited by the system generation program to create the run.be file for a new version. Once created, this file is copied to each backend.

The system generation software source code and use of shell scripts were designed to follow good software engineering practices. These practices include thorough documentation, modularity, a header file for constants, and the absence of global variables.

Flow diagrams can be found in Appendix D for the overall algorithm and three of the most complex functions. A listing of the source code can be found in Appendix E. Sample shell scripts called by the system generation software are listed in Appendix G.

F. A SUMMARY

As stated in the design requirements earlier in this chapter, a system generation management tool should be easy to use, do what it is expected to do, be reliable, and follow good programming principles. The software I have developed meets these requirements as demonstrated by the ease of use by first time users, doing what the user expected it to do, recovering from user errors often with specific error or warning messages, and setting the example for good programming principles in its code, users' manual, and documentation. Performance and ease of use of MDBS have improved since the system generation management tool has been installed.

IV. USING THE SYSTEM GENERATION SOFTWARE

The system generation software can start up, reconfigure backends, distribute processes when needed, and assist in the creation of new versions. A step by step description of how to use the software is contained in Appendix B.

A. TO START MDBS

The system generation software has simplified the start-up process. To start MDBS, all you type is **begin**. The system-generation program will look up the last MDBS configuration used and display the configuration. After answering a few questions, most of which only require '**y**' or '**n**' for a response, the system starts up in the configuration you wanted.

B. TO RECONFIGURE THE BACKENDS

The number of backends can depend upon the size of a database you plan on running or the testing you want to perform. Changing which backends to include in the next run of MDBS is fairly simple. After typing **begin** or **begin -r**, the system generation software will display on the screen the current version name, controller, and backends in the configuration. To change the configuration, answer '**y**' to the question "`Do you want to reconfigure the Back Ends? (y/n)`". For each available backend, you will answer '**y**' or '**n**' indicating which backends you want in the new configuration. After you have responded to each backend and are satisfied with the new configuration, it is saved in the configuration file. If you do not choose any backends, then system generation program will reread the configuration file and use the previous configuration.

C. FOR CONFIGURATION MANAGEMENT

After modifying code in MDBS which affects any backend process, you need to recompile the affected code and copy the resulting executable files to each backend. Start the system generation management program by typing '**begin -d**'. The program will ask if you need to recompile the backend code. If you need to recompile, it will initiate the

compiling[1] and then copy the compiled code to each backend. As long as you have created the applicable directory on each backend (or previously used the system generation software to create them) then all the backend executable code will get copied to the directory on each backend corresponding to the current version name.

D. FOR VERSION CONTROL

Creating a new version involves a fair amount of planning. Someone will copy the applicable portions of code in the MDBS hierarchy into new directories containing the name of the new version to be worked on. Then he modifies, as necessary, the make files to reflect the directory structure. The system generation and software management tool program is started with '**begin -v**' option. System generation program will ask for the new version name. After you enter and confirm the new version name, the program will change the version name in the configuration file. My system generation program will verify that the include file was updated to reflect the current version name as the default version name. The the system generation program will verify that the new version name can be found in the current directory.

[1] Compiling should be done before running the system generation software management tool program since you will be unable to verify the compilation was successful before the program copies code to the backends. If compilation was unsuccessful, you will have to exit this program to fix any errors then recompile.

V. AREAS FOR FURTHER STUDY WITH THE MDBS

The purpose of MDBS is to demonstrate to academia, business, and government that building a database computer which is multilingual and scalable (with multibackends) is not only possible, but a practical solution to the problem of data sharing between heterogeneous databases. As with any research project, there are many ways that project can be improved. In this chapter I have listed a few areas of improvement and directions for further research. The topics are listed in their order of importance as I see them.

A. UPGRADING EQUIPMENT

As time passes, computer technology improves. The current MDBS is implemented on Integrated Solutions, Inc. Optimal V24 model. These machines, commonly referred to as ISIV's, were purchased during the early 1980's. As discussed in Chapter I, the speed of a central processor has increased several times and disk drives capacity has increased at least 10 times since the early 1980's. Advances in technology and time have a cost. Machines become out of date. Machines also wear out and the hardware requires more frequent repairs. There comes a time when the repair costs exceed replacement costs. The MDBS will be getting SUN workstations to replace the ISIV's. Anyone who has upgraded equipment, even for "upward compatible" computers, knows that one cannot just copy the files and recompile. Every once in a while you might get lucky and not have to debug, but more often than not there will be differences between machines which will require debugging the system.

One complication with multiple hardware configurations is how to handle future upgrades to MDBS. If software development is shifted completely to db11 as controller with db12 and db13 as backends, how will any improvements be transferred to the two machines on loan?[1] Any future changes can not be distributed by making tape backups and sending them via mail to be loaded on db1 and db2. Since there are differences between the

[1] db1 and db2 at Pt Mugu and SUPSHIPS San Francisco.

new and old hardwares, the changes will have to be duplicated and tested on db3, db4, or db7 (with the same ISIV architecture), then copied to tape before sending to db1 and db2. This process has a fair amount of overhead.

B. INTEGRATING THE TWO MAIN VERSIONS

There are two versions of the Multilingual Multimodel Database System at the Naval Postgraduate School (NPS). One version[2] uses the same machine, db3, as the controller and the backend. This version implements the five language interfaces discussed in Chapter II. The second version[3] uses one machine as the controller and one or more machines as backends. The second version works reliably for the attribute based data model and sporadically for the other data models.

The Object-Oriented Database Model has been added to the version on db3. Since that version supports multilingual, not multibackend parallel database operations, the Object-Oriented code should be integrated into the multibackend version on db8. The other language interfaces must also be examined. The multibackend version on db8 has a significant amount of multilingual code. Since the db8 version does not work consistently with the languages implemented, the multilingual code should be compared with the db3 multilingual code with functions consistently. Improvements to the db8 code should be made so there is at least one reliable multilingual *and* multibackend system.

C. IMPROVING EXISTING MODULES

The Daplex, or functional, database language does not work on MDBS. Much effort was put into this project; however, the task has not been completed. Work on the functional data model has resumed. The functional model can be used with artificial intelligence applications. Having a functioning functional database model would be an asset to the MDBS.

---

2. VerE.6 in the cs4322 account on db3.
3. greg in the mdbs account on db8.

The Object Oriented Data Model as implemented on MDBS resembles the Relational model too closely. More work on the OODM must be done to make it truly object oriented. It currently has limited inheritance implemented. Implementing full inheritance and the covering property are under development.

D. THE USER INTERFACE

Many systems exist with a menu driven user interface. Menu driven displays are adequate for some purposes, but when they call for non multiple choice answers, the user may need additional information displayed on the screen, possibly in a different window. Anyone who has used Windows on a personal computer, or X Windows on a workstation can admit that their productivity increases when they have the ability to access references and get helpful information in other windows while they are doing their main work in another window.

Pop up windows also make the windows environment more user friendly since information can be seen when you need it. When you no longer need the information, you can remove the window, or the system can be programmed to remove the window.

On the older ISIV equipment, there is a primitive windows environment available, but the MDBS software does not automatically utilize any window features. The user has to explicitly open windows that he needs. Having multiple windows is very useful when running MDBS. At start up and any time during system operation, a user can find out if the multiple processes are running. Multiple windows can be used to find the exact names for databases, query files, template files, and other files. Even these uses of multiple windows require the user to have some knowledge of the system. One improvement would be to display the valid choices in a window instead of relying on the user to remember the exact file name, database name, or even the full path name of the file. Another improvement would be to have a separate process which monitors the main processes and lets the user know all is well and what died if all is not well with the multiple system processes. This can be accomplished by X Windows on workstations using the UNIX operating system. A

recent graduate from the Naval Postgraduate School wrote a windows environment interface for MDBS. This interface was written using transportable applications environment (TAE)[4] Plus and could be used as a template to start such an interface.

E. IMPLEMENTATION OF MORE CROSS-MODEL ACCESS

As displayed in Figure 1, there are only two cross-model database mappings implemented, Load using Hierarchial (DL/I), Query with Relational data language and Load using Object Oriented Data Language, Query with Relational Data Language. Studying and implementing more cross-model accessing can make MDBS harder to overlook when academia, business, and government attempt to solve the growing problem of heterogeneous databases.

F. REDISTRIBUTION OF DATA IN AN EXISTING DATABASE

A commercial system with multiple backends must be capable of error recovery. If one backend fails then the system should be able to be reconfigured automatically or with minimal human intervention. One database application is a bank. A database for a bank cannot be interrupted due to a failure of one backend for hours or days while the data (on the backends) is reconstructed. To solve this type of problem on MDBS, three major areas must be addressed, duplication of base data on backends, and redistribution of the base data, and database integrity.

When MDBS data is stored on multiple backends, the data is partitioned. When the number of backends changes, the partitioning also changes. Some system monitoring software should be able to either copy the back up data for the backend which failed and copy it to a backend not in use[5] or to back-up (store) all existing data on a common device (e.g. magnetic tape, multiple disk drives) and then reload the database on the new configuration. On MDBS this could be done by storing each data partition on at least two

---

4. TAE is a registered trademark of the National Aeronautics & Space Administration.
5. To keep the same database partitioning which would maintained by having the same number of backends in the new configuration.

backends and having one or more backends as ready spares which a monitoring system could use if a backend in use fails. Database integrity is discussed in (Quantock, 1989).

G. UPGRADING EXISTING CODE FROM C TO C++

Some people say that the writing is on the wall for C code. C++ is an Object-oriented language and can do much more than C could. Object-oriented databases should be implemented as part of a multilingual, multimodel database system. Future code for the multilingual, multimodel database system should be written in a higher level language than C, namely C++.

H. MULTIPLE USER AND MULTIPLE DATABASE

MDBS is a research database computer and does not have everything that is expected of a commercial database. An area of development not being worked on is allowing MDBS to have multiple users and multiple databases loaded simultaneously. A commercial database system must be able to handle multiple users simultaneously and should be able to handle multiple databases concurrently.

VI. PROBLEMS

MDBS is not a commercial product. It has been worked on mostly by students earning their masters degree in computer science at the Naval Postgraduate School. Since MDBS is not meant to be commercial strength, there is less emphasis placed on the optimal user interface, sufficient source code comments, or on error recovery and more upon showing that something new can be done with the system. Each language interface implements a subset of the commands for that language. The first language interface mapping was from relational (SQL) to the attribute based data language and model. This was the best written and documented schema transformation and query translation. The subsequent language interfaces were based on the design used by the relational model and language. Although there have been several paid systems programmers to get the multibackend system running and occasionally to make it better, there are currently no paid staff responsible for MDBS. Unfortunately, the system stops unexpectedly. Some reasons for the system to stop are data entry error[1], a software error, and a hardware failure.

A. RETRIEVE - COMMON

The kernel data language uses the retrieve common command to perform a join on attribute-value pairs (similar to equi-join in the relational model). During the retrieve common process, back-ends must communicate with each other. The inter process (IP) buffer has a size of 1000 characters. Messages sent between processes can have up to 1425 characters. The operating system uses a flushing mechanism for messages longer than 1000 characters. Some indications are that this mechanism is not given enough time to complete the flushing task before the arrival of the rest of the message or the arrival of the next message. As a result, the messages are damaged. The damaged messages causes errors when calculating virtual addresses for the data (Hammond, 1992, p.42). It was proposed

[1] Made by the database user with a corresponding lack of error recovery built into the source code by a programmer.

that an upgrade of the operating system[2] might solve these problems. This remains to be seen as the system is being ported over to the more powerful workstations.

A second problem with retrieve common is a result of backends having to communicate amongst one another. For commands other than retrieve common, backends only communicate with the front end. The front end will either broadcast a message to one or to all backends. In response each backend will send one or more messages to the front end. If there are several retrieve commons being performed, or if one retrieve common has a large set of data to join, the amount of inter-backend communications increases. This increased traffic on the ethernet causes the number of collisions to increase. As a result, inter-machine communications are slowed down. This delay is severe enough that during the bench marking, the retrieve common was not included in the commands tested (Hsiao, 1991, p. 54). This communications related problem may be solved by using a protocol similar to the token ring protocol. Using a fiber distributed data interface (FDDI) would solve the communications bottleneck two ways. First, the data rate is much higher than the ethernet (10 vs. 100 Mega bits per second)[3]. Second, the protocol does not permit collisions.

B. SOFTWARE RELIABILITY

Since there is not error recovery, the system locks up at unexpected times. You can be using a "perfectly" coded language interface and the system can still crash or lock up. There are many possible problems. One problem could be due to the design of the Ethernet (broadcasting messages can collide and not be detected). If a message gets lost, the controller can wait indefinitely expecting a message which will never be received. If the backend sent the message, then it will not send it again. This problem could also be due to

---

2. The ISIV's were installed with ISI 4.2 Berkeley Software Distribution (BSD) Release 3.07. The operating system was upgraded to 4.3 BSD Virtual VAX 11 Version 490145 Rev B dated October 1987. When the movement to Sun Workstations is completed, MDBS will be using UNIX Sun OS 4.1.1.

3. If messages can not be sent to multiple recipients then a network of 10 backends would only benefit from the lack of collisions, not from the higher data rate.

how sockets are handled by the operating system (BSD 4.3) as discusses in the previous section.

MDBS is meant to show that a multimodel, multilingual, multibackend database system can be implemented. Once government and industry realize the strengths of both the multilingual and multibackend, then the idea could spread to a commercial strength product which would better handle data entry errors and remove most--if not all--software errors.

VII. CONCLUSIONS

As more demands are placed on computing resources, it is vital that database systems be designed which get the most out of each computing dollar. Heterogeneous databases exist in most large organizations. Their existence causes a lack of effective data sharing, duplication of data, and wasted computing resources. One way to solve these problems is to use a kernel data model and language to store all databases. Users can view, load, and modify a database in whatever data model and language they would like to use. The Multilingual, Multimodel, Multibackend Database Supercomputer located at the Naval Postgraduate School has demonstrated the feasibility of such a system.

A. WHAT I HAVE ACCOMPLISHED

Running multi-process systems is hard enough on multiple machines, but adding multiple versions on top of this becomes a difficult task. The more we can automate, the less a supervisor, software developer, or user will have to know or do to run the system. There is a definite need for automation where ever possible. Human beings do not need to waste time remembering exact command names and machine specific peculiarities. Innovative ideas in the computer industry have continued to increase productivity of computers and users. Other ideas which have improved the user interface and thus productivity are icons, pull down menu's, and a consistent use of color. The lack of a command line, although intimidating to more experienced command line users, shifts the burden of remembering correct commands from the user to writing a better interface to the programmer of that interface.

After using the MDBS I realized its merits and its possibilities for solutions to real world problems. I also realized that the learning curve to use the system is long and steep. One of my goals is to make the MDBS easier to learn and easier to use. With the code I have written (see Appendix E), I have automated running MDBS. Getting the system to run does not solve all the problems. Knowing how the system operates is also important in easing the learning curve.

Documentation is very important and cannot be stressed enough. The lack of comments today in a program that you know inside and out will become a huge barrier for anyone--including yourself--from looking at that code in the future and being able to quickly understand not only what it does, but how the program does it! I have thoroughly documented my code and have gathered information to explain the overall setup of the file system which supports MDBS. Within that I have described many of the temporary and permanent files that make MDBS operate. Together with the User's Manual (Appendix A to Bourgeois 1993), which explained how to use the multilingual aspect of MDBS, it is now much easier for someone new to learn the MDBS from a database user's perspective and from a systems programmers viewpoint.

B. SOME FUTURE AREAS OF RESEARCH RELATED TO THIS WORK

Areas for possible future research are discussed in the previous chapter. The areas which need to be worked on are the upgrading of equipment, improving the reliability of the software, object oriented interface, and more cross-model accessing capability. Upgrading of equipment to Sun4 workstations is in progress which may improve the reliability of existing software. A full-time staff or contractor needs to work on, test, and improve the performance of the multibackend version of MDBS[1] to at least the performance of the single backend version of MDBS[2].

The object oriented interface needs to be less relational and more object-oriented. The interface needs to be integrated into the multibackend version of MDBS which will be done once the system is ported to the Sun4 workstations.

And finally the cross-model accessing capability. I believe this is the greatest strength of MDBS. If academia, business, and government would see the benefits of a Federated Database System using a kernel data model and language, there could be great steps toward data sharing and the eventual reduction in the number and sizes of databases in the world

---

1. The current version name is 'greg' which is an improvement on version '7'.
2. Implemented on db3 as 'VerE.6'.

today given a constant amount of information to store. This reduction would result in a tremendous savings in the cost of maintaining existing databases.

APPENDIX A. FILE ORGANIZATION OF MDBS

This appendix presents an overview of the directory structure and important files used to operate MDBS. First we review the basic hierarchy of the DATABASE software files starting at the Version's Directory which is maintained on the controller (currently ISIV8).

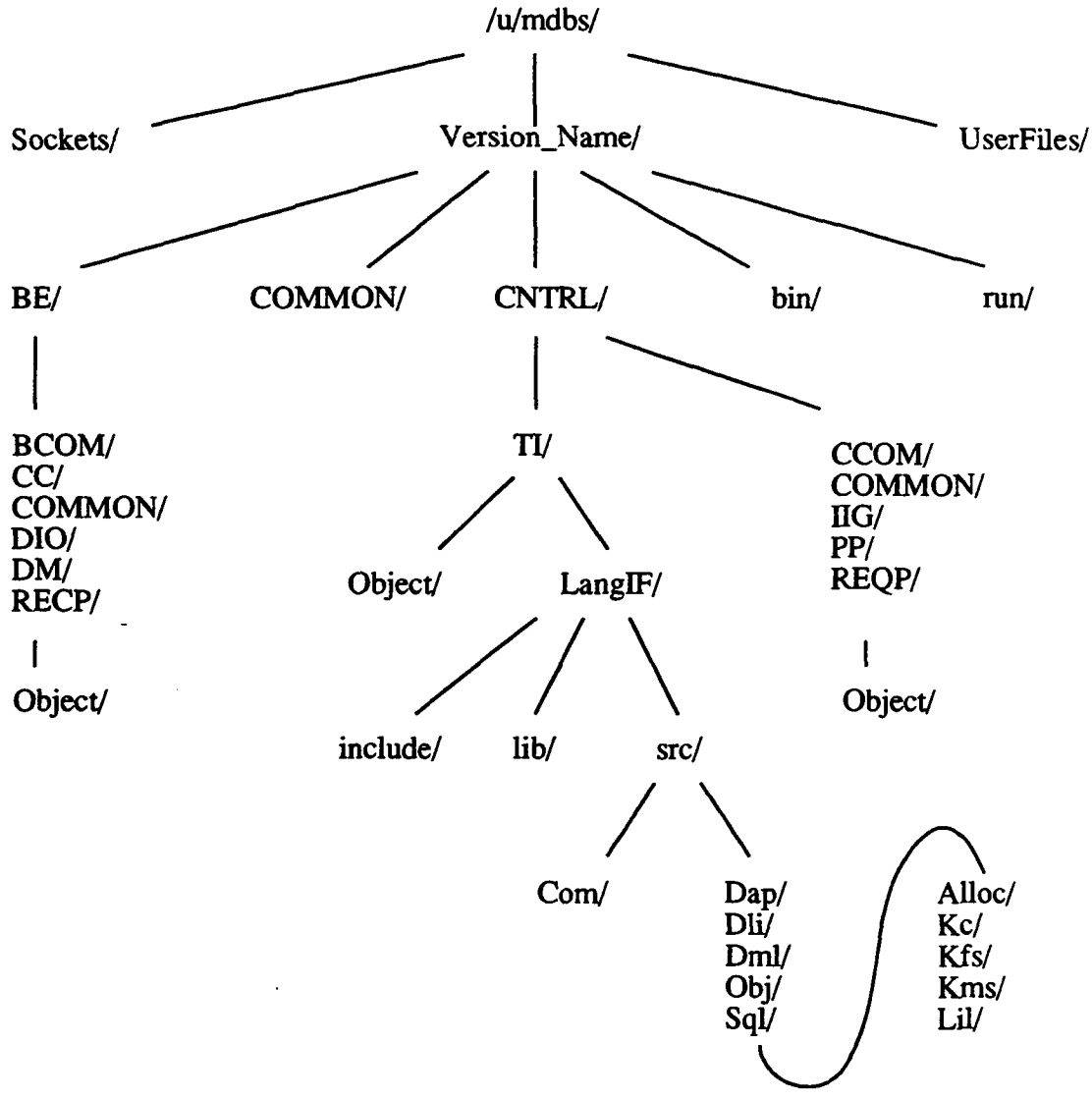


Figure 7. Directory Structure of the MDBS files on the controller

A. CONTROLLER

BE: Directory containing:

the following 6 sub-directories which create the 6 Back End Processes:[1]

COMMON: source files common to all Back End processes.

CC: Concurrency Control.

DIO: Data Disk Input/Output.

DM: Directory (meta disk) Management - where records are put.

RECP: Record Processing.

BCOM: Back End Communications - Machine to machine.

the executable code for each process (bget, bput, cc, dio, dirman, recp).exe[2]

CNTRL: Directory containing:

the following 6 sub-directories which create the 6 Controller Processes:[3]

COMMON: source files common to all Controller processes.

CCOM: Control Communications - cget and cput processes.

RECP: Request Preparation - ABDL Parser

PP: Post Processing -

IIG: Insert Information Generation - determine in what track to put a record

TI: Test (user) Interface - source and libraries for each non kernel Data Manipulation Language except ABDL

the executable code for each process (cget, cput, iig, pp, reqp, ti).exe[4]

---

[1] Each has its own Object/ directory used during the compilation process for include and object files.

[2] The .exe extension is used to differentiate this version from the cs4322 version which uses the .out extension.

[3] See footnote 1.

[4] See footnote 2.

COMMON: Directory containing:

source common to Back End and Controller

bin: Directory containing:

shell scripts, binaries, and makefile used to copy or move files and data, view data and meta data, and compile any and all parts of the system.

constants - display meta table sizes (source in Version_Name/BE/DM)

cpcount - copy a file for a given number of bytes. Use to copy meta and data disks to other BE (source in Version_Name/BE/DIO)

cpydisks - script using cpcount, NOT to be used on CONTROLLER

cpytobe - copy BE processes to backend (not used)

disp - display dynamic meta table (source in Version_Name/BE/DM)

gdb - generate test database of arbitrary size from template file (usage: gdb D.t D[.r])(source in Version_Name/CNTRL/TI)

makeall - shell script which calls make in the appropriate directory for each of the 12 processes

makefile - can be called with 4 options. make all - calls makeall; make auto - compiles main.c; make clean - deletes all object files; make dist - creates a tar'd, compressed backup file of the distribution.

rectag - print out formatted data from data disk (source in Version_Name/BE/DIO)

zero - writes 0's to a file; used to erase meta and data disks before loading a new database (source in Version_Name/BE/DIO)

configure.h - include file for main.c

main.c - source for automatic reconfiguration, redistribution, and execution of MDBS

run: Directory for:

files (1) created by the controller during execution of MDBS storing information about the database and (2) needed to start and restart MDBS.

.Syntax -

.TransFile -

.config.db - version name and list of backend names in current configuration. Used by main (specifically from Version_Name/bin/main.c program).

.curr_file - temporary file used by some language interfaces

.exe.awk - pattern matching for awk command in stop.<controller> (e.g. stop.db8)

.output - last query result displayed to the screen.

.qry_file - the last displayed create/query (SQL) or dbd/request (DL/I) file.

trace/ - directory for trace files from the 5 controller processes

<process name>.tr - trace for most recent execution of that process

database list files: - (.dap.dbl, .hie.dbl, .net.dbl, .obj.dbl, .sql.dbl) one file for each language, one database name per line. These files are created each time MDBS is run. Each file is created from its respective .*dbs.dat files.

database data: - (.dapdbs.dat, .hiedbs.dat, .netdbs.dat, .reldbs.dat) one file for each language, a listing of all information needed to create a database catalog (.*.cat) file for each known database. These files are updated when a database schema is added via creation or translation/transformation from another schema.

database catalogs: - (.DTH.cat, .DTH2cat, etc.) one file for each database listed in each .*.dbl file. Information is created at run time (probably when the specified language interface is requested; could be at start of ti.c) by reading the corresponding .dat file.

shell scrips: all called by Version_Name/bin/main.c

start.cntrl - start the first 5 of 6 controller processes

master.run.be -a template used to create run.be for a new version of software

run.be - latest run.be created from master.run.be

stop.<machine_name> - stop processes remaining from last run of database

zero.<back_end_name> - erase data from back end meta and data disks

zero.<controller_name> - erase files storing data of previous database

/CNTRL/TI/LangIF - Test Interface, Language Interface

include: files included in .c files at compile time (e.g. .dcl, .ext, .h, .def)

lib: archived files. One per Data Language (sql.a, dli.a, dml.a, dap.a obj.a).

src: source code for each data manipulation language (DML). Sub directories are:

Com: three files used by each data language.

Sql: Relational Data Language (SQL) -

Dli: Hierarchial Data Language -

Dml: Network Data Language -

Dap: Daplex Data Language -

Obj: Object Oriented Data Language -

Each Data Language has up to 5 sub directories:

Alloc: Allocation of memory for various data structures.

Kc: Kernel control - communicates between ABDL, Kms, and Kfs.

Kfs: Kernel formatting system - collects and formats data from Kc for display in Lil

Kms: Kernel mapping system - parse queries generated by Lil. Sends to Kc.

Lil: Language Interface Level - menus for user. Communicates with Kms, Kc, and Kfs.

The following directories and files must be present on the CONTROLLER within $HOME or the login directory:

Version_Name/ described above. As a minimum, you need the 6 executable files for the controller processes and all of the run directory to start the system.

UserFiles/ formats of files are described in the User's Manual, chapter IX (Bourgeois, 1993, pp. 84-88).

DB_NAME.d - descriptor file. Places restrictions on values and ranges of values of attributes. Used in clustering.

DB_NAME.r - mass load file. For Relational and Attribute Based Models.

DB_NAME.t - template file. Structures of the attribute-based database.

DB_NAME#n - ABDL query list. Also known as Traffic Unit.

DB_NAME<sql, dml, dli, obj, dap>db - schema file - needed by system to create a new database unless you do it interactively. This file contains enough information to generate the descriptor and template files

OTHER - files such as DB_NAMEsqlreq create table and SQL select

Sockets/ 11 connections between Backend and Controller

CC=

DM=

IIG=

P_PCLB=

RECP=

TI=

DIO=

G_PCLC=

PP=

P_PCLC=

REQP=

.<each control executable in CNTRL/>.pid - (not used) process ID of last run of that process

.alias - aliases to facilitate navigation in the directory structure as well as the **begin** MDBS alias which must be updated if the current version is moved or renamed. At the time of this writing, **begin** is defined as: alias begin 'cd /u/mdbs/greg/run; main'.

.cshrc

.login

B. BACK END

The second part of the file sructure of Multibackend Database Supercomputer (MDBS) to be described is the back end file structure. Again starting with the directory structure of each back end, see Figure 8.

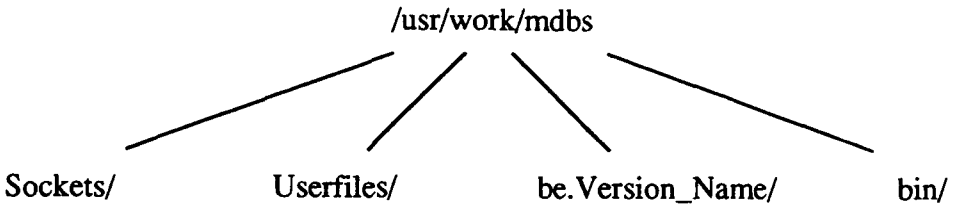


Figure 8. Directory Structure on Backend Machines

EACH Back end has the following files and directories:

/usr/work/mdbs - login directory containing:

.(bget, bput, cc, dio, dirman, recp).pid - process IDs for last run (not used)

be.Version/ -- one directory per version

- 6 process executable file names (bget, bput, cc, dio, dirman, recp).exe

  6 process_name.tr files for stdout and stderr while running

  run.be*

UserFiles/ - copies or links to files with the same name as in UserFiles/ on the controller. Both files must exist on every back end for every database used on the system.

DB_NAME.r

DB_NAME.t

bin/ - copies of files or links to files in Version_Name/bin may be included. The following must exist for the automatic reconfiguration and start-up program to work

stop.cmd - from Version_Name/bin

exe.awk - from Version_Name/run

zero - from Version_Name/bin

Sockets/ - 6 connections between Backend and Controller

CC=

DIO=

DM=

G_PCLB=

P_PCLB=

RECP=

APPENDIX B. HOW TO START AND RECONFIGURE MDBS

The following is a description of what you need to do to start and run the Multi-backend Database Supercomputer (MDBS) as automated by my program.

Initial Conditions:

Login as mdbs to the controller, currently db8, open one window or use the console for the controller to run MDBS. To monitor the processes and check the meta and data disks, you should rxterm or create another window again to the controller and once for each of the Back Ends you plan to use. This can be done by being logged in to each of the applicable consoles and/or have multiple windows open on your terminal, one for each back end and two for the controller.

A. TO START MDBS

At the UNIX prompt on the controller type:

start or **begin**

When you do this you are changed to directory greg/run and the program configure begins execution. If the start or begin aliases have not been updated at the last version change, then an error message will be displayed stating that the version name "<name>" from .config.db is not in current path. If this message occurs, either change the alias or manually cd <Version_name>/run; main.
The following will be displayed once the configure program starts execution

```
Welcome to MDBS, today is    Fri Feb 19 12:54:05 PST 1993


Check the time each file was last compiled:

-rwxrwxr-x  1 mdbs        83649 Nov  5 12:58 ../BE/bget.exe
-rwxrwxr-x  1 mdbs        83649 Nov  5 12:58 ../BE/bput.exe
-rwxrwxr-x  1 mdbs       246357 Nov  5 12:57 ../BE/cc.exe
-rwxrwxr-x  1 mdbs        89208 Nov  5 12:57 ../BE/dio.exe
-rwxrwxr-x  1 mdbs       375581 Nov  5 12:58 ../BE/dirman.exe
-rwxrwxr-x  1 mdbs       335559 Nov  5 12:59 . /BE/recp.exe

-rwxrwxr-x  1 mdbs        91811 Nov  5 12:52 ../CNTRL/cget.exe
-rwxrwxr-x  1 mdbs        91811 Nov  5 12:52 ../CNTRL/cput.exe
-rwxrwxr-x  1 mdbs        56680 Nov  5 12:52 ../CNTRL/iig.exe
-rwxrwxr-x  1 mdbs        56971 Nov  5 12:53 ../CNTRL/pp.exe
-rwxrwxr-x  1 mdbs        84382 Nov  5 12:53 ../CNTRL/reqp.exe
-rwxrwxr-x  1 mdbs       488244 Nov  5 12:56 ../CNTRL/ti.exe

There should be 12 files listed, if not you need to recompile

Do you need to recompile any executable and/or copy the 6
 executable files to each Back End
(bget, bput, cc, dio, dirman, recp.exe)? (y/n)
```

Most of the time you will enter **n** to this question.

Next you will be shown the name of the current version, which machine is the controller and which are the back ends.

```
The Current Configuration is:

Version Name:   greg

Controller:     db8

3 Back Ends:
                db3
                db4
                db7

 WARNING:  All data will be lost if you reconfigure

 Do you wish to reconfigure the Back Ends? (y/n)
```

Normally you would enter **n** so that you will use the same back end configuration. Reconfiguring is discussed in the next section.

If you do not reconfigure, then you will be asked if you want to use the last data base loaded on the system. Unless you know which database was used last and you want to use that database, enter **n**. If you answer **y**, all data and meta data will be erased.

```
 Do you wish to use current database? (y/n)
```

If you enter n, the following will be displayed for each back end:

```
Zeroing backend meta disk on back end, db4...
File to zero = /dev/sd1c
File size = 105638400
Bytes to zero = 1000000
Bytes written...
     819200
    1000000
Zeroing backend data disk on back end, db4...
File to zero = /dev/sm0c
File size = 418775040
Bytes to zero = 100000
Bytes written...
     100000
```

Then for the controller, the following will appear with a possible "No match" or two.

```
Removing CINBT and IIG AT tables on controller, db8...
Removing sql, hie, and catalog files
```

Now that the configuration is determined, you are given the opportunity to not run MDBS. This option is here in case someone wants to reconfigure, stop unwanted jobs, recompile code, or copy code to back ends without actually running MDBS.

```
Do you wish to run the Multi Model, Multi Lingual,
              Multi Backended Database System? (y/n)
```

Entering **n** will exit you from this program and return you to the UNIX shell. Entering **y** will cause the following steps to occur:

Configure will check the file .config.db to see the last combination of backends. Any process running on the controller or backend will be terminated (with the shell scripts named: stop.<machine_name>). For the configuration in this example and no duplicate processes still running, the following will be displayed.

```
stopping processes on back end db3
no processes to kill on db3
stopping processes on back end db4
no processes to kill on db4
stopping processes on back end db7
no processes to kill on db7
stopping processes on db8, the controller
no processes to kill on db8
```

Then 5 of the 6 processes on the controller are started in the background.

```
starting 5 of 6 controller processes on db8...

[1] 2510
[2] 2511
[3] 2512
[4] 2513
[5] 2514
```

For each back end, six processes are started.

```
Running backend on db3...
[1] 1358
[2] 1359
[3] 1360
[4] 1361
[5] 1362
[6] 1363
Running backend on db4...
[1] 2302
[2] 2303
[3] 2304
[4] 2305
[5] 2306
[6] 2307
 Running backend on db7...
[1] 808
[2] 809
[3] 810
[4] 811
[5] 812
[6] 813
```

At this point, the user interface begins.

Note that the 6th process on the controller, ti.exe, does not start right away. It must communicate with the other 5 processes and usually dies if any of the other processes are not ready to communicate via the Sockets.

You can verify using each window (or terminal as applicable, described above in the initial conditions) that all 6 processes are running on each back end.

For each Back End type:

When you are logged in (or rlogin) to that machine: **ps** or **ps | grep exe**
When you have a window on another machine then: **rsh <Back End Name> ps**

To verify the 6 processes are running on the controller type:

ps or **ps | grep exe**

If any process is not running on any machine in the current configuration, then you must exit the main program on the controller by typing 'x' at the first prompt. Otherwise, continue to run MDBS in accordance with the User's Manual which can be found in (Bourgeois, 1993).

B. TO RECONFIGURE MDBS

Initial Conditions are described at the beginning of this Appendix.

At the UNIX prompt on the controller type:

start -c or **begin -c**

Follow the instructions for To START MDBS. If you forget the -c then answer y to the following WARNING and question.

```
WARNING:  All data will be lost if you reconfigure

Do you wish to reconfigure the Back Ends? (y/n)
```

Enter y to each backend that you want in the new configuration. Once you confirm your choice, the .config.db file will be updated and each machine in the new configuration will be zeroed for loading a new database.

```
Indicate which Back Ends you want in the configuration
by responding y/n as each Back End is listed:

                db3 ? y

                db4 ? n

                db6 ? n

                db7 ? y

                db9 ? n

The Current Configuration is:

Version Name:   greg

Controller:     db8

2 Back Ends:
                db3
                db7
Is this configuration satisfactory? (y/n)
```

Once you confirm your choice with a y, the .config.db file will be updated and each machine in the new configuration will be zeroed for loading a new database. A sample zero for one back end and for the controller is shown in the previous section, To START MDBS. After zeroing is complete, you will be given the option to continue or exit MDBS.

```
Do you wish to run the Multi Model, Multi Lingual,
                Multi Backended Database System? (y/n)
```

Enter y to start up the system with the new configuration. For a description of what happens next, refer to the previous section, To START MDBS.

C. TO DISTRIBUTE BACKEND EXECUTABLE CODE

Initial Conditions:
- The same as described at the beginning of this appendix, plus
- you are in the directory for the current version since the code will be copied to the directory corresponding to the current version on each back end.

To tell the configure program that you want to distribute executable back end code to a new or existing back end. At the UNIX prompt enter:

start -d or **begin -d**

If you forget to use the -d option, you can answer **y** to the first question as described above in START MDBS.

```
Welcome to MDBS, today is  Fri Feb 19 14:48:48 PST 1993


Check the time each file was last compiled:

-rwxrwxr-x  1 mdbs        83649 Nov  5 12:58 ../BE/bget.exe
-rwxrwxr-x  1 mdbs        83649 Nov  5 12:58 ../BE/bput.exe
-rwxrwxr-x  1 mdbs       246357 Nov  5 12:57 ../BE/cc.exe
-rwxrwxr-x  1 mdbs        89208 Nov  5 12:57 ../BE/dio.exe
-rwxrwxr-x  1 mdbs       375581 Nov  5 12:58 ../BE/dirman.exe
-rwxrwxr-x  1 mdbs       335559 Nov  5 12:59 ../BE/recp.exe

-rwxrwxr-x  1 mdbs        91811 Nov  5 12:52 ../CNTRL/cget.exe
-rwxrwxr-x  1 mdbs        91811 Nov  5 12:52 ../CNTRL/cput.exe
-rwxrwxr-x  1 mdbs        56680 Nov  5 12:52 ../CNTRL/iig.exe
-rwxrwxr-x  1 mdbs        56971 Nov  5 12:53 ../CNTRL/pp.exe
-rwxrwxr-x  1 mdbs        84382 Nov  5 12:53 ../CNTRL/reqp.exe
-rwxrwxr-x  1 mdbs       488244 Nov  5 12:56 ../CNTRL/ti.exe

There should be 12 files listed, if not you need to recompile.

Do you need to recompile before copying to the back ends? (y/n)
```

If the code has not been modified since the last compilation, then enter **n**. Entering **y** will execute a 'make all' on the system which may take several minutes to complete. It is much safer to execute 'make all' from outside of this program so that you can verify the compilation was successful before copying any code to the back ends, as described in Appendix F.

After all 6 BE processes are copied to every back end defined in the file, configure.h., the configuration is displayed and you are asked if you want to reconfigure.

```
The Current Configuration is:

Version Name:    greg

Controller:      db8

3 Back Ends:
                 db3
                 db4
                 db7

 WARNING:  All data will be lost if you reconfigure

 Do you wish to reconfigure the Back Ends? (y/n)
```

From here on refer to the previous section, TO START MDBS.

D. TO CHANGE VERSIONS

note: disk space is limited on almost any system. Also the more versions in existence, the harder it is to integrate changes on older versions into the newer version(s). Minimize the number of copies and versions of code.

Initial Conditions:
- The same as described in To START MDBS plus
- Replication of the entire[1] controller directory tree has been completed from an existing version.[2]

After changing the following alias so that it changes directory[3] to the new version, at the UNIX prompt on the controller, type:

start -v or **begin -v**

This will display similar information as described above in To START MDBS except the first prompt will be asking you to confirm that you want to change the version.

```
The Current Configuration is:

Version Name:    greg

Controller:      db8

3 Back Ends:
                 db3
                 db4
                 db7

WARNING:(1) changing the version should only be done for major changes.
        (2) Please consult Dr. Hsiao or all project members first.
        (3) This program MUST be run from the new version directory
              e.g. from directory /u/mdbs/VERSION/run.

       pwd == /u/mdbs/greg/run


Do you still want to change the current version? (y/n)
```

---

1. Note that a partial copy of the code can be made as long as the associated makefile(s) for that copy are modified to find the files from another version that were not copied or modified. When developing the Object-Oriented Data Language Interface, only the CNTRL/LangIF directory was copied on db3 to the rich directory. The controller process ti.out was the only process of twelve to be modified. The other eleven processes from cs4322/VerE.6 were used by simply modifying the run command in mdbs/rich/run directory
2. For example, if you wanted to copy the entire version 7 from db4, you could execute the following command from the login directory of the (new) controller:
   cp -r /@db4/u/mdbs/7 NEW_VERSION
3. The configure program will check the present working directory for the new version name, if it is not present, the program will display an error message and stop.

If you enter **y**, you will be asked to enter and verify the name of the new version

```
Enter the new version name: greg
Please verify the new version name to be: greg
Is this correct? (y/n)(q/Q to Quit) y
```

You can exit the version change process by entering **q** or **Q**. You can enter **n** to re-enter the new version name. When you enter **y**, you will be asked if the Back End directory be.NEW_VERSION has been set up.[4]

```
pwd includes new version, updating .config.db
Have Back Ends been set up for this version? (y/n)
```

After answering **y** or **n**, the current configuration is displayed with the updated Version name.

```
The Current Configuration is:
Version Name:    thesis
Controller:      db8
3 Back Ends:
                 db3
                 db4
                 db7
 WARNING:  All data will be lost if you reconfigure
 Do you wish to reconfigure the Back Ends? (y/n)
```

From this point on, this program continues as previously described in To START MDBS.

WARNING: If you did not run the configure program from the directory of the new version, then this program will terminate with an Error message and instructions telling you what needs to be done to continue.

---

[4] Set-up includes creating the directory be.VERSION, creating a new run.be file for the VERSION, copying the new run.be to be.VERSION, and copying the executable code for the 6 Back End processes to be.VERSION.

E. TO LOAD MDBS ON A NEW MACHINE

Initial Conditions:
- mdbs account exists in /usr/work/mdbs directory.
- the current mdbs machines and account can access the new machine via "rcp dbn:/" where n is the number of the new machine.
- configure.h has been updated to include the new machine as either a back_end[] or as the controller and main.c which includes this file has been recompiled
- the CPU and operating system are identical[5]

As a CONTROLLER
Copy[6] the entire directory tree[7] which is described in Appendix A

Look at pcl.def file which lists two socket addresses and the controller name used by mdbs..

As a BACK END
mkdir /u/mdbs/bin and copy from another back end:
stop.cmd, awk.exe, and zero. If the new machine is not identical to the one you copied from, then you must reompile the zero file on the new machine using the code from the BE/DIO subdirectory
mkdir /u/mdbs/be.<CURRENT_VERSION>
copy executables and run.be from the same directory on another back end.

mkdir /u/mdbs/Sockets

mkdir /u/mdbs/UserFiles

5. If they are not then a detailed evaluation of the source code for system calls must be done. Refer to the masters thesis of MAJ Stan Watkins to be completed by September 1993.
6. The entire code can be copied from another machine, say db8, by starting at the login directory of the new machine and typing: rcp -r db8:/u/mdbs/greg NEW_VERSION
7. It is possible to only copy the 6 controller processes, and the NEW_VERSION/run files; however, you will need more if you plan on doing any revisions to the code on the new machine.

F. SAMPLE ERROR MESSAGES

1. Invalid Option

If you forget what the options are or what they do, enter an invalid one such as start -h or begin -q. You will get the following response:

```
Welcome to MDBS, today is  Tue Mar  2 12:56:47 PST 1993


unrecognized option h,  valid options: [-c -v -d]

-c  change Back End CONFIGURATION
-d  copy (DISTRIBUTE) executable code to Back Ends
-v  change VERSION name
```

2. Incorrect new version name

If you change the version name (e.g. to '7') in a directory that does not contain the version name you will see (after you confirm the new version name):

```
ERROR:  7 not found in pwd.
        .config.db should only be updated in /mdbs/7/run directory.

        .config.db not updated.
```

3. Incorrect version for pwd

If you try to run mdbs with the version in .config.db not in pwd (present working directory), you will get the error message displayed below.

```
 Do you wish to run the Multi Model, Multi Lingual,
               Multi Backended Database System? (y/n) y


ERROR:  version name "7" from .config.db
        is not in current path: /u/mdbs/greg/run

        If version name is not correct, restart with -v option
        to correct version name.

        If version name is correct, then you are running main
        from the wrong directory.  You must either
                run main from /mdbs/7/run    or
                change begin / mdbs / start alias to cd to /mdbs/7/run

        When changing versions, remember to verify default_version is
        correct in configure.h.  If a change is made,
        recompile main.c then copy main to /mdbs/7/run/main

        Exiting configure program.
```

4. Include file out of date

If you try to run a new version from the correct directory, but forget to change the initialization of the global variable 'default_version' in configure.h to equal the new version name, you will get the following error message:

```
ERROR:  default_version "greg" does not match
        current_version "andy"
        If current_version is correct,
                Go to /mdbs/andy/bin directory
                In configure.h correct default_version
                compile main.c by typing 'make auto'
                makefile will automatically copy main to run/main

        If current_version is not correct, restart with -v option
        to correct version name.

        Exiting configure program.
```

5. Missing file configuration file

If the .config.db file is missing from the current directory, you will see the following WARNING:

```
WARNING:  unable to read .config.db

Configuring for 1 Back End:  db4
                Controller:  db8
                Version:     mdbs
```

6. Missing or Incorrect process names

If a file is missing or you did not change the make file in /u/mdbs/ VERSION/___ (CNTRL & BE) to name the 12 executable files to end in ".exe: then you will see:

start.cntrl: ../CNTRL/<process_name>.exe: not found

...

for each back end

run.be: bget.exe: not found

APPENDIX C. ON-LINE DOCUMENTATION

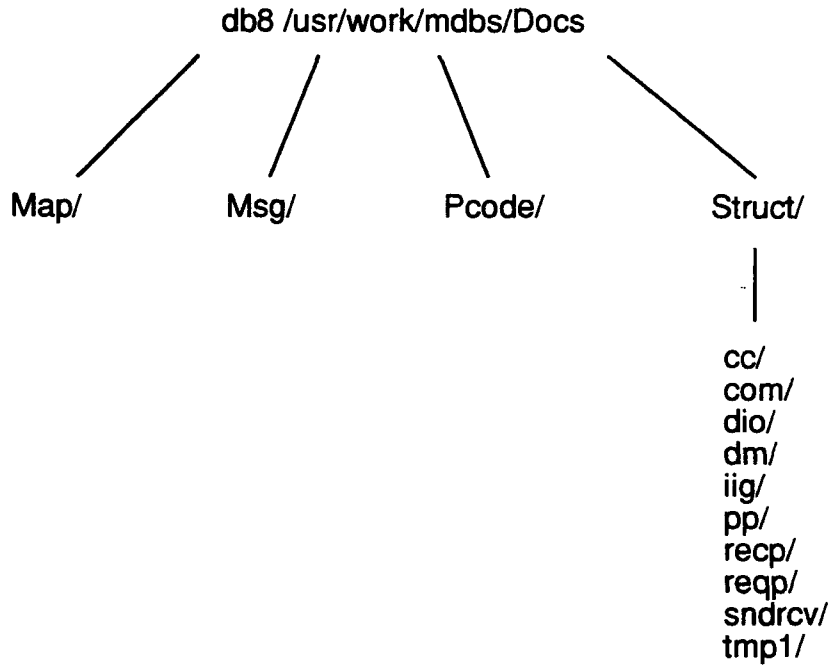


Figure 9: Documentation of MDBS design and structures

The documentation files are mostly text files with occasional troff formatted files which contain formatting commands and can be read as text. There are some diagrams in the sub-directories of Struct which are text files.

Map/

A file for some of the 12 processes listing the function names, source file where function can be found, and the purpose of the function. The latest revision was done in May of 1988.

cc.map - concurrency control
common.map -
com_msg.map -
dio.map - disk i/o process
dm.map - directory management process
iig.map - process iig;
pp.map - post processing
recp.map - record processing
reqp.map - request preparation
ti.map - test interface

Msg/

A file for each type of message used for inter process communications. Each file contains type, sender process, receiver process, source file, and purpose.

guide - contains information on the format of the files in this directory.
toc - contains the table of contents for this directory.

Pcode/

Pseudo code explaining the general sequence of events performed by each process. The code is not complete, and was probably written in 1988.

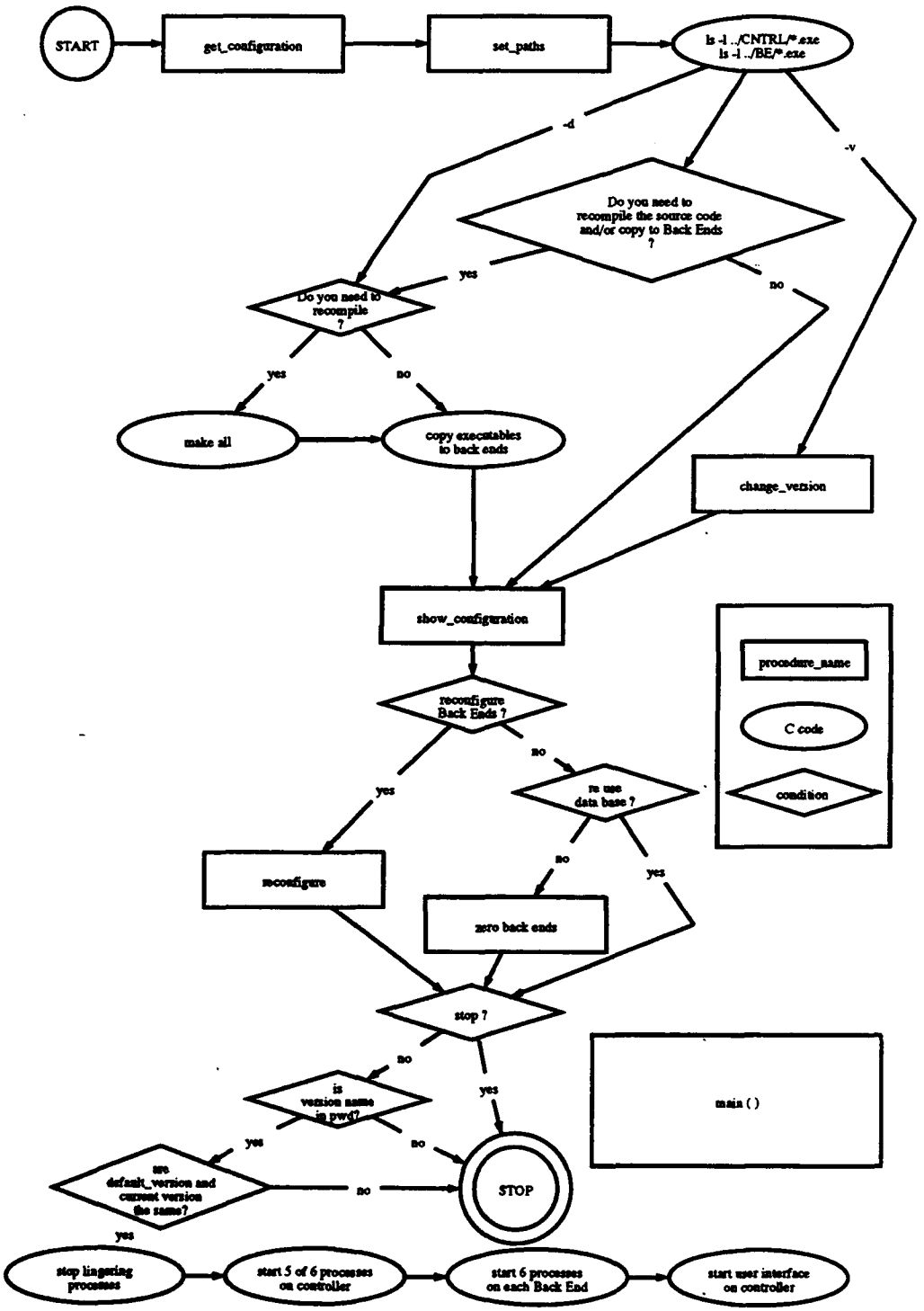
Communication.pc
cc.pc
common.pc
dio.pc
dm.pc
grecp.pc
iig.pc
recp.pc
reqp.pc
ti.pc

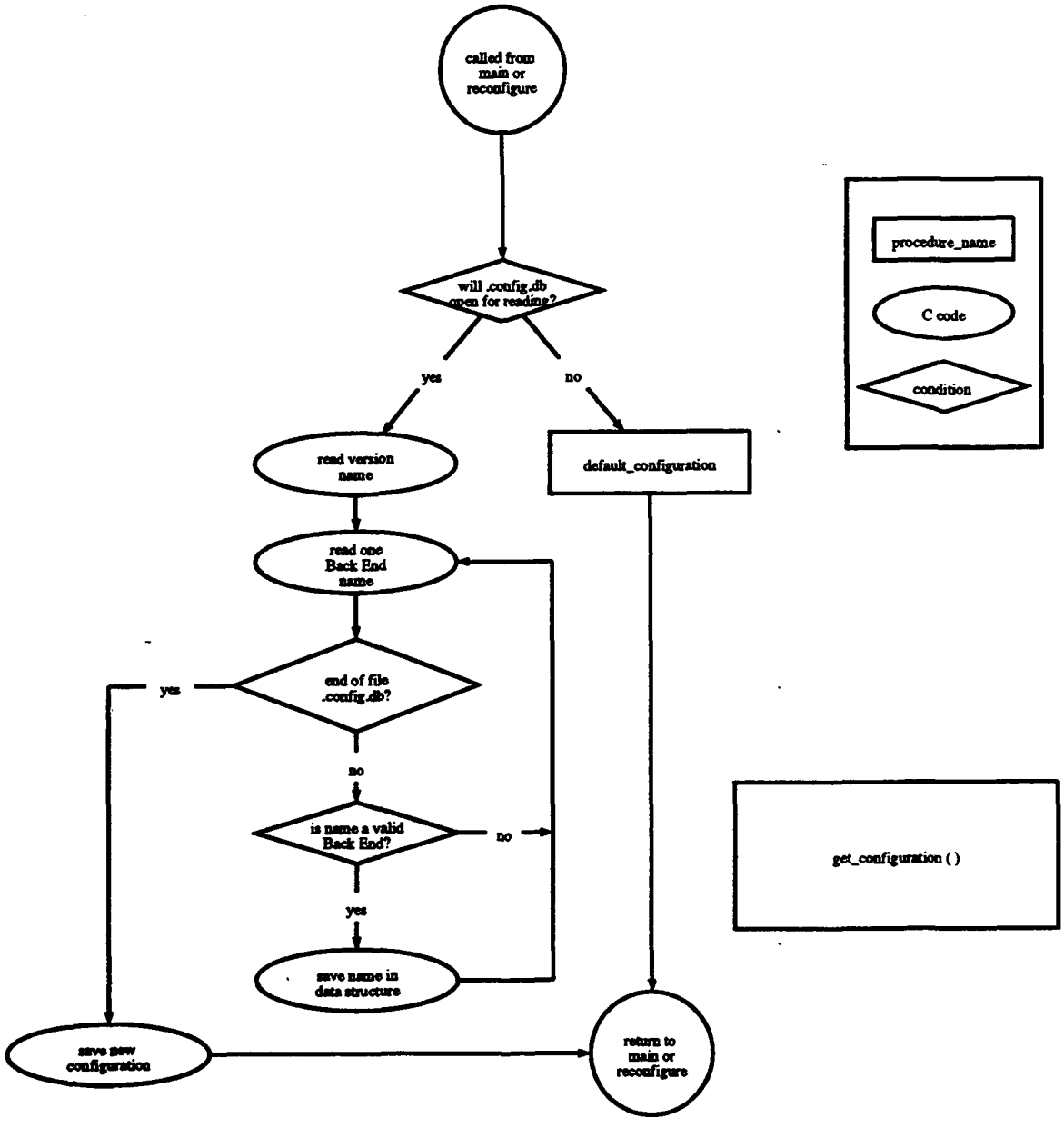
Struct/

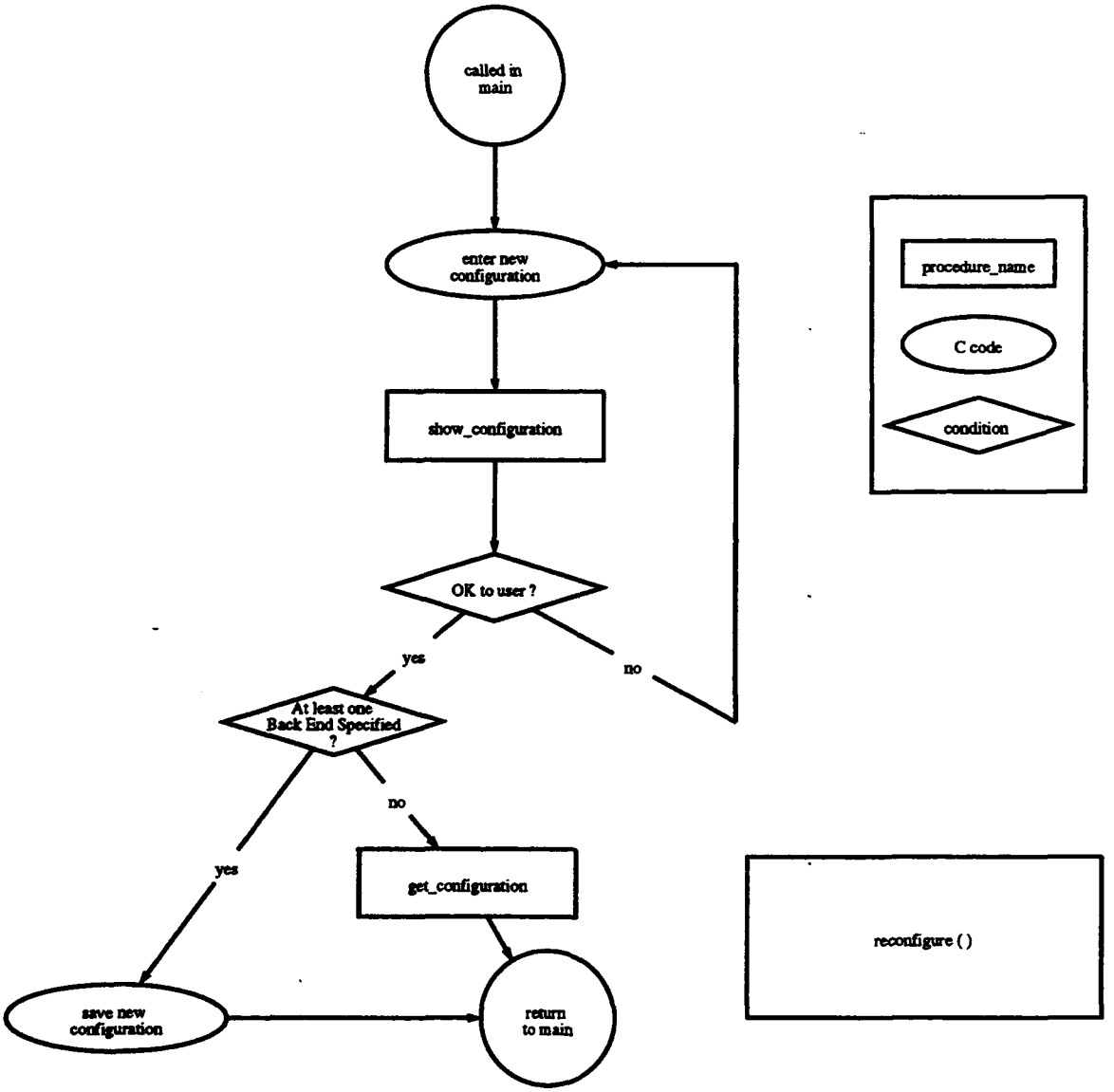
cc/
com/
dio/
dm/
iig/
pic
pic1
pic2
pp/
print_them* - a useful shell script to print all the *.s files
recp/
reqp/
sndrcv/
tmpl/

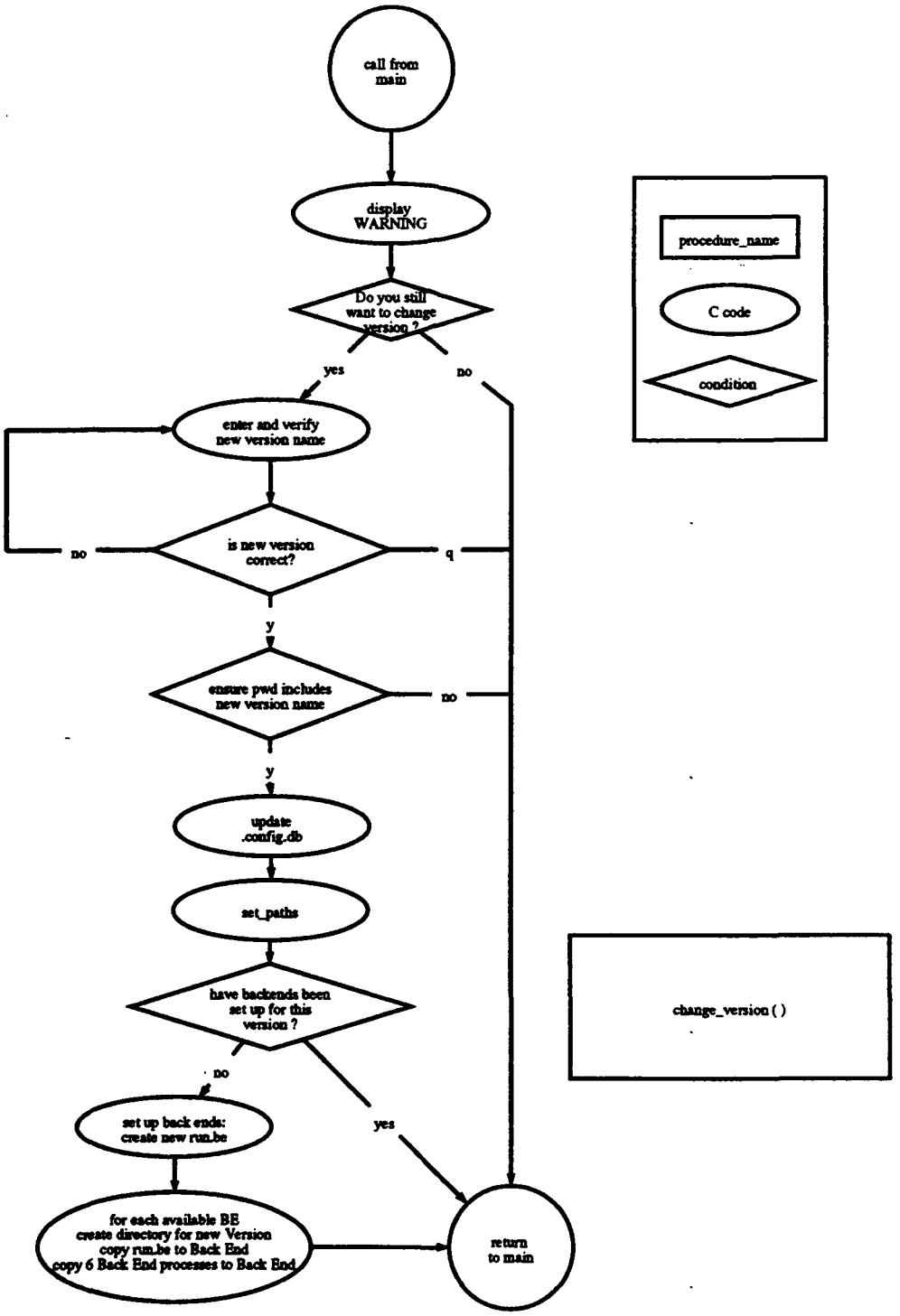
APPENDIX D. FLOW CHARTS FOR SYSTEM GENERATION SOFTWARE

The following four pages describe the control flow of the software I developed to make MDBS a more manageable system. The first page is an overview of the whole program. It describes the main() procedure. The following three pages show more detail of three of the functions called by the main procedure. The three functions are get_configuration(), reconfigure(), and change_version().


START
get_configuration
set_paths
ls -l ../CNTRL/*.exe
ls -l ../BE/*.exe
-d
-v
Do you need to recompile the source code and/or copy to Back Ends ?
yes
no
Do you need to recompile ?
yes
no
make all
copy executables to back ends
change_version
show_configuration
reconfigure Back Ends ?
no
yes
re use data base ?
no
yes
reconfigure
zero back ends
stop ?
no
yes
is version name in pwd?
yes
no
are default_version and current version the same?
yes
no
STOP
stop lingering processes
start 5 of 6 processes on controller
start 6 processes on each Back End
start user interface on controller
procedure_name
C code
condition
main ( )


called from main or reconfigure
will .config.db open for reading?
yes
no
read version name
defauk_configuration
read one Back End name
end of file .config.db?
yes
no
is name a valid Back End?
no
yes
save name in data structure
save new configuration
return to main or reconfigure
procedure_name
C code
condition
get_configuration ( )


called in main
enter new configuration
show_configuration
OK to user ?
yes
no
At least one Back End Specified ?
yes
no
get_configuration
save new configuration
return to main
procedure_name
C code
condition
reconfigure ( )


call from main
display WARNING
Do you still want to change version ?
yes
no
enter and verify new version name
is new version correct?
no
q
y
ensure pwd includes new version name
no
y
update .config.db
set_paths
have backends been set up for this version ?
no
yes
set up back ends: create new run.be
for each available BE create directory for new Version copy run.be to Back End copy 6 Back End processes to Back End
return to main
procedure_name
C code
condition
change_version ( )

APPENDIX E. SOURCE CODE FOR SYSTEM GENERATION SOFTWARE

The following code is found in Version_Name/bin and is compiled by being in that directory on db8 or db11 and typing 'make auto'. Auto is short for automatic reconfiguration and start-up of the Multibackend Database Supercomputer (MDBS).

The first file, configure.h is the include file for main.c on db8. The second file, configure.h is the include file for main.c on db11. The third file is main.c, the same on all machines.

```
/*-------------------------------------------------------------------------
| UNIT_NAME     | CONFIGURATION                                    configure.h
|               | include file for main.c
|               |                    stored in directory:   current_version/bin
| PROGRAM       |
| DESCRIPTION   |    Start the Multi-Lingual Parallel Distributed
|               | Database program with varying number of Back Ends.
|               |
|               | files needed to run main.c:
|               |           ===>  on Controller  <===
|               | in current_version/run/:
|               |    zero.dbn       : shell script which uses rsh to call the
|               |                       zero command on the named back end
|               |                       number 'n' to zero meta & data disks
|               |    stop.dbn       : shell script which uses rsh to call the
|               |                       stop.cmd command on the named back end
|               |                       number 'n' or execute on controller
|               |                       to stop 6 processes
|               |    start.cntrl    : shell script which starts 5 of 6
|               |                       controller processes in background
|               |    master.run.be  : template used to create back end run.be
|               |    .config.db     : current version name followed by a
|               |                       list of back ends in current configuration
|               |    main           : this code compiled
|               |
|               | in current_version/CNTRL/:
|               |    all 6 controller processes
|               |
|               |          ===>  on each Back End  <===
|               | in /u/mdbs/be.current_version/:
|               |    *.exe      : all 6 back end processes
|               |    run.be     : shell script to start 6 Back End processes
|               |    trace files for all 6 processes (will be created)
|               |
|               | in /u/mdbs/bin/:
|               |    zero       : C program to erase memory
|               |                          (source in current_version/BE/DIO)
|               |    stop.cmd   : shell script to kill last 6 pid's run
|               |    exe.awk    : called by stop.cmd, pattern match *.exe in ps x
|               |
|               | The code in this program is represented pictorially, see thesis
|               |
| INPUTS        | stdin, .config.db file
| OUTPUTS       | stdout
| CREATED       | 19 October 1992
| MODIFIED      | many times through May 1993
| AUTHOR        | Andrew P. Meeks
| PROJECT       |
| PURPOSE       | AUTOMATE the START-UP process of the Multi-Modal,
|               |      Multi-Back ended, Multi-Lingual Database System
| System        | UNIX
| Compiler      | cc
-------------------------------------------------------------------------*/
/* IF HARDWARE CONFIGURATION CHANGES (equipment added, fixed, removed, breaks,
              etc) you must review the following 7 lines of code for change */

/* enum all_isivs  { db1, db2, db3, db4, db5, db6, db7, db8, db9 };
char *isiv_names[] = {"db1", "db2", "db3", "db4", "db5", "db6", "db7", "db8", "db9"};
*/

    /*   must have at least MAX_BACK_ENDS "    " defined          */
    /*   the length of the blank strings must be at least as long as the
             longest back end name                                 */
    /*   the following should be declared inside main(), but the compiler
             will not allow "automatic aggregate initialization"
             unless declared was in the header file                */
char *current_BackEnds[] = {"    ", "    ", "    ", "    ", "    "};
```

```
/* uncoment next 3 lines for use with db1 at Pt Mugu:
char *be_names[]         = {"db1"};
enum Back_Ends             { db1 };
enum Back_Ends default_be = db1;
   be sure to comment out or delete subsequent duplicate declarations        */

/* uncoment next 3 lines for use with db2 at SUPSHIP San Francisco:
char *be_names[]         = {"db2"};
enum Back_Ends             { db2 };
enum Back_Ends default_be = db2;
   be sure to comment out or delete subsequent duplicate declarations        */

char *be_names[]         = {"db3", "db4", "db6", "db7", "db9"};
enum Back_Ends             { db3,   db4,   db6,   db7,   db9 };
enum Back_Ends default_be = db4;

char *controller = "db8";          /* to change, need to
                                       (1) update pcl.def include file
                                       (2) recompile executable code        */

               /* enum Back_Ends and all_isivs can be used
                  to index be_names[] and isiv_names[]                       */

/* change 5 to 1 for use with single back end systems db1 and db2             */
#define MAX_BACK_ENDS  5
#define MAX_STRING    64    /* used for ALL non constant string declarations */

/* IF VERSION IS CHANGED they you must update the following line accordingly */
char default_version[MAX_STRING] = "mdbs";   /* constant */


/* The following two lines must correspond to output of "make all"  in
   /u/mdbs/VERTION/bin directory.   Actual files created in the makefiles
   of each process                                                           */
char *BE_process[]          = {"bget.exe",  "bput.exe",   "cc.exe",
                               "dio.exe", "dirman.exe", "recp.exe"};

char *CNTRL_process[]       = {"ti.exe ", "cget.exe", "cput.exe",
                               "iig.exe", "pp.exe",   "reqp.exe" };
       /* "ti.exe " includes a space since it is called with a parameter    */
int N_PROCESSES = 6;

#define FALSE 0
#define TRUE  1
#define ERROR 1
#define NO_ERROR 0

#define NO  0
#define YES 1

/*      STATUS of equipment 11/6/92:

          db1    Pt. Mugu
          db2    SUPSHIP San Francisco
          db3    back end,   working
          db4    back end,   working
          db5    no meta or data disk
          db6    no data disk
          db7    back end,   working
          db8    controller, working
          db9    no data disk
 */
/* eof:  configure.h */
```

```
/*-------------------------------------------------------------------------
| UNIT_NAME     | CONFIGURATION                                    configure.h
|               | include file for main.c
|               |                    stored in directory:   current_version/bin
| PROGRAM       |
| DESCRIPTION   |    Start the Multi-Lingual Parallel Distributed
|               | Database program with varying number of Back Ends.
|               |
|               | files needed to run main.c:
|               |           ===>  on Controller  <===
|               | in current_version/run/:
|               |    zero.dbn       : shell script which uses rsh to call the
|               |                        zero command on the named back end
|               |                        number 'n' to zero meta & data disks
|               |    stop.dbn       : shell script which uses rsh to call the
|               |                        stop.cmd command on the named back end
|               |                        number 'n' or execute on controller
|               |                        to stop 6 processes
|               |    start.cntrl    : shell script which starts 5 of 6
|               |                        controller processes in background
|               |    master.run.be  : template used to create back end run.be
|               |    .config.db     : current version name followed by a
|               |                        list of back ends in current configuration
|               |    main           : this code compiled
|               |
|               | in current_version/CNTRL/:
|               |    all 6 controller processes
|               |
|               |          ===>  on each Back End  <===
|               | in /u/mdbs/be.current_version/:
|               |    *.exe      : all 6 back end processes
|               |    run.be     : shell script to start 6 Back End processes
|               |    trace files for all 6 processes (will be created)
|               |
|               | in /u/mdbs/bin/:
|               |    zero       : C program to erase memory
|               |                        (source in current_version/BE/DIO)
|               |    stop.cmd   : shell script to kill last 6 pid's run
|               |    exe.awk    : called by stop.cmd, pattern match *.exe in ps x
|               |
|               | The code in this program is represented pictorially, see thesis
|               |
| INPUTS        | stdin, .config.db file
| OUTPUTS       | stdout
| CREATED       | 19 October 1992
| MODIFIED      | 27 April 1993 (by S.Watkins for new Sun hardware)
| AUTHOR        | Andrew P. Meeks
| PROJECT       |
| PURPOSE       | AUTOMATE the START-UP process of the Multi-Modal,
|               |      Multi-Back ended, Multi-Lingual Database System
| System        | UNIX
| Compiler      | cc
-------------------------------------------------------------------------*/
/* IF HARDWARE CONFIGURATION CHANGES (equipment added, fixed, removed, breaks,
               etc) you must review the following 7 lines of code for change */

/* enum all_isivs  { db11, db12, db13 };
char *isiv_names[] = {"db11", "db12", "db13"}; */

    /*   must have at least MAX_BACK_ENDS "    " defined          */
    /*   the length of the blank strings must be at least as long as the
             longest back end name                                 */
    /*   the following should be declared inside main(), but the compiler
             will not allow "automatic aggregate initialization"
             unless declared was in the header file                */

char *current_BackEnds[] = {"    ", "    ", "    ", "    ", "    "};
char *be_names[]         = {"db12", "db13"};
```

```
enum Back_Ends              { db12,    db13 };
enum Back_Ends default_be = db13;

char *controller = "db11";          /* to change, need to
                                       (1) update pcl.def include file
                                       (2) recompile executable code        */

               /* enum Back_Ends and all_isivs can be used
                  to index be_names[] and isiv_names[]        */

#define MAX_BACK_ENDS  2
#define MAX_STRING    64    /* used for ALL non constant string declarations */

/* IF VERSION IS CHANGED they you must update the following line accordingly */
char default_version[MAX_STRING] = "greg";

char current_version[MAX_STRING];  /* read in from .config.db file           */

char BE_path[MAX_STRING];     /* see set_path (); "/u/mdbs/be.VERSION/";      */
char CNTRL_path[MAX_STRING]; /* see set_path (); "/u/mdbs/VERSION/CNTRL/";    */

/* The following two lines must correspond to output of "make all"  in
   /u/mdbs/VERTION/bin directory.   Actual files created in the makefiles
   of each process                                                           */
char *BE_process[]          = {"bget.exe",  "bput.exe",   "cc.exe",
                               "dio.exe", "dirman.exe", "recp.exe"};

char *CNTRL_process[]       = {"ti.exe ", "cget.exe", "cput.exe",
                               "iig.exe", "pp.exe",   "reqp.exe" };
        /* "ti.exe " includes a space since it is called with a parameter    */
int N_PROCESSES = 6;

#define FALSE 0
#define TRUE  1
#define ERROR 1
#define NO_ERROR 0

#define NO  0
#define YES 1

/*      STATUS of old ISIV equipment 930427:

          db1    Pt. Mugu
          db2    SUPSHIP San Francisco
          db3    back end,   working
          db4    back end,   working
          db5    no meta disk
          db6    no data disk
          db7    back end,   working
          db8    controller, working
          db9    no data disk
 */
/* eof:  configure.h */
```

```
/*-------------------------------------------------------------------------
| UNIT_NAME     | CONFIGURE                                  main.c
|               |             file stored in directory:    VERSION/bin
| PROGRAM       |
|  DESCRIPTION  | start the Multi-Lingual, Parallel, Distributed
|               | Database program with varying number of Back Ends.
|               |
|               | GENERAL OUTLINE:
|               | check if called with -c, -v, -d options  check_for_flags ()
|               | read previous configuration              get_configuration ()
|               | set controller and back end paths        set_paths ()
|               | display configuration                    show_configuration ()
|               | if -v option used
|               |   get new version name                   change_version ()
|               | if -c option not used, ask if user wants to reconfigure
|               |   WARNING:  will erase database
|               |   if YES or -c option used
|               |      erase previous database data        zero ()
|               |      get  new configuration              get_configuration ()
|               |      save new configuration
|               |   if NO  and  -c option NOT used
|               |      ask if user wants to use present database
|               |      if NO
|               |        erase previous database data      zero ()
|               | kill all lingering processes             system (stop.machine)
|               | start 5 Controller processes             system (start.cntrl)
|               | start all Back End processes             system (rsh run.be)
|               | start ti on controller                   system (ti.exe #)
|               |
|               | NOTES:
|               | Version can ONLY be changed if main called with -v option
|               | Controller is defined in "configure.h" to be db8
|               | Available Back Ends are defined in "configure.h"
|               |
|               | The code in this program is represented pictorially, see thesis
|               |
|               | WARNING:  revise configure.h with each version change
|               |
| INPUTS        | stdin, .config.db file
| OUTPUTS       | stdout
| CREATED       | 19 October 1992
| MODIFIED      | many times through March 1993
| AUTHOR        | Andrew P. Meeks
| PROJECT       | MDBS
| PURPOSE       | get multi-model, multi-backended, multi-lingual database
|               |     system to work
| System        | UNIX
| Compiler      | cc
-------------------------------------------------------------------------*/
#include <strings.h>
#include <stdio.h>
#include "configure.h"
```

```
/* ---------------------------------------------------------------------
 * CHECK_FOR_FLAGS  evaluate command line parameters
 *
 *     parameters:  argc, argv     (in)       same as in main()
 *                  pNew_ver       (in out)   whether -v option used
 *                                            used for version change
 *                  pNew_config    (in out)   whether -c option used
 *                                            used for configuration change
 *                  pDist_code     (in out)   whether -d option used
 *                                            used to copy code to BE's
 *     you can only used one option per '-' sign
 *
 *     return       FALSE  if no invalid options used to call this program
 *                  ERROR  if an invalid option was used
 */
int check_for_flags (argc, argv, pNew_ver, pNew_config, pDist_code )
int argc;
char *argv[];
int  *pNew_ver, *pNew_config, *pDist_code ;
{
    char *pCH;                      /* pointer to parameter string   */
    int  temp = 1;

    if (temp < argc)                /* at least one parameter used   */
    {
        while (temp < argc)         /* another parameter to process  */
        {
            pCH = argv [temp];
            if (*pCH == '-')        /* first character must be a '-' */
            {
                pCH ++;
                switch (*pCH)       /* remember which option(s) used */
                {
                    case 'c' :  *pNew_config = TRUE;
                                break;
                    case 'v' :  *pNew_ver    = TRUE;
                                break;
                    case 'd' :  *pDist_code  = TRUE;
                                break;
                    default  :  printf ("unrecognized option %c,  ", *pCH);
                                printf ("valid options: [-c -v -d]\n\n");
                                printf ("-c  change Back End CONFIGURATION\n");
                                printf ("-d  copy (DISTRIBUTE) executable ");
                                printf ("code to Back Ends\n");
                                printf ("-v  change VERSION name\n\n");
                                return (ERROR);
                                break;
                }
            }
            temp ++;                /* process next option           */
        }
    }
    return (FALSE);
}   /* end check_for_flags () */
```

```
/* ------------------------------------------------------------------
 * NOT_IN_PWD return TRUE if string (Version name) in PWD string
 *
 *  parameters:  pwd             (in) current working directory
 *               current_version (in) version name
 *
 *  return:      TRUE   if current_version found in pwd string
 *               FALSE  if current_version not found in pwd string
 */
int not_in_pwd (pwd, current_version)
char *pwd;
char *current_version;
{
    char *pch  = pwd;       /* pointer to current portion of pwd        */
    int length = strlen (current_version);

    while (*pch)
    {
        pch++;              /* increment pointer to character after '/' */
        if (strncmp (pch, current_version, length))
        {
            /*  scan path for next '/' but not past end of string  */
            while (*pch != '/' && *pch)
                ++pch;
        }
        else
        {   /* found current_version in pwd */
            return (FALSE);
        }
    }
    return (TRUE);

} /* end not_in_pwd () */



/* ------------------------------------------------------------------
 * SHOW_CONFIGURATIONdisplay version, controller, and back ends to be used
 *
 *  parameters:  n_BackEnds         (in)  the number in current Configuration
 *               current_BackEnds[] (in)  strings of each back end name
 *               current_version    (in)  name of version in .config.db
 *
 *  return:      no information
 */
int show_configuration (n_BackEnds, current_BackEnds, current_version)
int n_BackEnds;
char *current_BackEnds[];
char *current_version;
{
    int i;                  /* array index                          */

    printf ("\nThe Current Configuration is: \n");
    printf ("\nVersion Name:\t%s \n", current_version);
    printf ("\nController:\t%s \n", controller);
    if (n_BackEnds == 1)
        printf ("\n1 Back End: \n");
    else
        printf ("\n%d Back Ends: \n", n_BackEnds);

    for (i=0; i< n_BackEnds; i++)
        printf ("\t\t%s \n", current_BackEnds[i]);
}
```

```
/* ------------------------------------------------------------------------
 * DEFAULT_CONFIG   configures database to have one back end
 *
 *  parameters:  pN_BackEnds        (out) number in default Configuration
 *               current_BackEnds[] (in)  strings of each back end name
 *               current_version    (in)  name of version in .config.db
 *
 *  return:      NO_ERROR if default configuration saved in .config.db
 *               ERROR    if unable to open .config.db
 */
int default_config (pN_BackEnds, current_BackEnds, current_version)
int *pN_BackEnds;
char *current_BackEnds[];
char *current_version;
{
    FILE *pFILE;           /* pointer to .config.db file      */

    /* .config.db does not exist or cannot be written to */

    printf ("WARNING:  unable to read .config.db\n\n");
    printf ("Configuring for 1 Back End:  %s\n", be_names[default_be]);
    printf ("\t\tController:  %s\n",              controller);
    printf ("\t\tVersion:     %s\n",              default_version);

    *pN_BackEnds = 1;

    /* initialize current_version  */

    strcpy (current_version, default_version);

    strcpy (current_BackEnds[0] , be_names[default_be]);

    if (pFILE = fopen (".config.db", "w"))
    {
        fprintf (pFILE, "%s\n", default_version);
        fprintf (pFILE, "%s\n", be_names [default_be]);
        fclose  (pFILE);
    }
    else
    {
        printf ("unable to write .config.db\n");
        return (ERROR);
    }
    return (NO_ERROR);
}   /* end default_config () */


/* ------------------------------------------------------------------------
 * GET_CONFIGURATIONread .config.db file from pwd
 *                      initialize the two parameters:
 *
 *  parameters:  pN_BackEnds         (out)  number of Back Ends in .config.db
 *               current_BackEnds[]  (in)   list of back end names
 *               current_version     (out)  name of version in .config.db
 *
 *  return:      NO_ERROR if valid .config.db exists upon exiting
 *               ERROR    if unrecoverable error in .config.db
 */
int  get_configuration (pN_BackEnds, current_BackEnds, current_version)
int  *pN_BackEnds;
char *current_BackEnds[];
char *current_version;
{
    char pch[MAX_STRING];  /* temporary pointer to string     */
    FILE *pFILE;           /* pointer to .config.db file      */
    int found;             /* boolean, if BE is a valid one   */
    int i;                 /* array index                     */
```

```
    if (pFILE = fopen (".config.db", "r"))
    {
        /* .config.db file is open                              */
        fscanf (pFILE, "%s", pch);

        /* initialize current_version
                        should be strncpy (,,MAX_STRING); */
        strcpy (current_version, pch);

        if (strcmp (default_version, current_version))
        {
            printf ("\nError:  configure.h file has a different version name ");
            printf ("than the default\n");
            printf ("\t  char *default_version =\"%s\";  ", default_version);
            printf ("vs. .config.db:  \"%s\" \n\n ", current_version);
            printf ("\tAny time the version is changed, default_version in ");
            printf ("configure.h \n\t   must be updated.\n");
            printf ("\tUpdate configure.h as needed and recompile main.c\n");
            printf ("\tThe files are found in %s/bin\n", current_version);
            /* return (ERROR); not used since exits later */
        }


        while (fscanf (pFILE, "%s", pch) != EOF)
        {
            /* ! feof (pFILE)  */

            strcpy (current_BackEnds[*pN_BackEnds], pch);
            found = FALSE;
            for (i = 0; i < MAX_BACK_ENDS; i++)
                if (strcmp (pch, be_names[i]) == 0)
                {
                    found = TRUE;
                    break;
                }
            if (found)
                (*pN_BackEnds)++;
            else
                printf ("ERROR, back end name %s in .config.db invalid\n", pch);
        }   /* end while */
        fclose (pFILE);

        if (*pN_BackEnds == 0)
            return (default_config (pN_BackEnds, current_BackEnds,current_version));

    }       /* if fopen .config.db */
    else
    {
        return (default_config(pN_BackEnds, current_BackEnds, current_version));
    }
    return (NO_ERROR);
}   /* end get_configuration () */
```

```
/* ------------------------------------------------------------------------
 * ZERO_BACKENDS    calls a shell command in pwd which calls zero on
 *                    each back end in current configuration
 *
 *  parameters:  n_BackEnds (in)    number of BackEnds in current configuration
 *               current_BackEnds[] list of back end names in current config
 *
 *  return:      no information
 */
int zero_backends (n_BackEnds, current_BackEnds)
int n_BackEnds;
char *current_BackEnds[];
{
    int i = 0;                /* array index                          */
    char pch[MAX_STRING];     /* temporary pointer to string          */

    for (;  i < n_BackEnds; i++)
    {
        /* for each back end being used, call shell scripts to      */
        /*                               zero meta and data disks   */

        sprintf (pch, "zero.%s", current_BackEnds[i]);
        system  (pch);
    }
    /* on controller, call shell script to remove files not needed */

    sprintf (pch, "zero.%s", controller);
    system  (pch);
}
```

```
/* ------------------------------------------------------------------------
 * SET_PATHS  sets the path of BE_path, CNTRL_path
 *
 *  parameters:  current_version (in)    name of current version
 *               BE_path         (out)   complete path name for directory
 *                                         containing executable BackEnd code
 *               CNTRL_path      (out)   complete path name for directory
 *                                         containing executable Control code
 *  return:      no information
 */
int set_paths (current_version, BE_path, CNTRL_path)
char *current_version, *BE_path, *CNTRL_path;
{
    /* initialize BE_path     to  /u/mdbs/be.VERSION/  */
    strcpy (BE_path, "/u/mdbs/be.");
    strcat (BE_path, current_version);
    strcat (BE_path, "/");

    /* initialize CNTRL_path  to  /u/mdbs/VERSION/CNTRL/  */
    strcpy (CNTRL_path, "/u/mdbs/");
    strcat (CNTRL_path, current_version);
    strcat (CNTRL_path, "/CNTRL/");
}
```

```
/* ------------------------------------------------------------------------
 * CHANGE_VERSION change version name and / or copy executable code to
 *                each working back end
 *
 *  parameters:  pN_BackEnds       (in)   number of BackEnds in current
 *                                          configuration
 *               current_BackEnds[]       list of BackEnd names in current config
 *               current_version  (in)    name of current version
 *               BE_path       (in out)   complete path name for directory
 *                                          containing executable BackEnd code
 *               CNTRL_path    (in out)   complete path name for directory
 *                                          containing executable Control code
 *  returns:     YES  if new version
 *               NO   if no change to version
 */
int change_version (pN_BackEnds, current_BackEnds, current_version,
                    BE_path,     CNTRL_path,       pwd)
int *pN_BackEnds;
char *current_BackEnds[];
char *current_version;
char *BE_path;
char *CNTRL_path;
char *pwd;
{
    char pch[MAX_STRING];              /* temporary pointer to string         */
    char new_version[MAX_STRING];      /* new version-name                    */
    int i;                             /* array index                         */
    int found = TRUE;                  /* boolean if pwd includes new version */
    FILE *pFILE;                       /* file pointer to .config.db file     */

    /* ask if you want to change version */
    printf ("\nWARNING:\t(1) changing the version should only be done for ");
    printf ("major changes.\n ");
    printf ("\t\t(2) Please consult Dr. Hsiao or all project members first\n");
    printf ("\t\t(3) This MUST be run from the new version directory\n");
    printf ("\t\t    e.g. from directory /u/mdbs/VERSION/run.\n\n");

    printf ("\tpwd == %s\n", pwd);

    printf ("\nDo you still want to change the current version? (y/n) ");

    scanf ("%s", pch);
    printf ("\n");
    if (*pch == 'y' || *pch == 'Y')
    {
        *pch = 'n';

        while ( (*pch == 'n' || *pch == 'N') && (*pch != 'q' || *pch != 'Q') )
        {
            printf ("\nEnter the new version name: ");
            scanf ("%s", new_version);
            printf ("\n");
            printf ("\nPlease verify the new version name to be: %s\n",
                                                        new_version);
            printf ("\nIs this correct? (y/n)(q/Q to Quit) ");
            scanf ("%s", pch);
            printf ("\n");
        }

        if (*pch == 'q' || *pch == 'Q')
            return (NO);

        strcpy (current_version, new_version);

        /* ----- should verify version name valid. ------ */
        /* -------- no /*&@^%!|$# characters ------------ */
        /* NOT IMPLEMENTED */
```

```
        /* verify that new version is in the current path */
        /* ---------------------------------------------- */
        if (not_in_pwd (pwd, current_version))
        {
            printf ("ERROR:  %s not found in pwd. \n", current_version);
            printf ("\t.config.db should only be updated in /mdbs/");
            printf ("%s/run directory.\n\n", current_version);
            printf ("\t.config.db not updated.\n");
            return (NO);
        }

        /* -------- rewrite .config.db --------    */
        /* ------------------------------------    */
        if (pFILE = fopen (".config.db", "w"))
        {
            printf  ("pwd includes new version, updating .config.db \n");
            fprintf (pFILE, "%s\n", current_version);

            for (i=0;  i < *pN_BackEnds; i++)
            {
                fprintf (pFILE, "%s\n", current_BackEnds [i]);
            }
            fclose (pFILE);
        }
        else
        {
            printf ("unable to write to .config.db\n");
            return (NO);
        }

        /*  change BE_path and CNTRL_path to include new version name   */
        set_paths (current_version, BE_path, CNTRL_path);

        /* edit master.run.be file and put in sample run.be file in pwd
           to be copied to each back end                                */

        printf ("\nHave Back Ends been set up for this version? (y/n) ");
        scanf ("%s", pch);
        printf ("\n");
        if (*pch == 'y' || *pch == 'Y')
        {
            return (YES);
        }

        sprintf (pch, "sed -e \"s/VERSION/%s/g\" master.run.be > run.be",
                      current_version);
printf ("EXECUTING: %s\n", pch);
        system (pch);

        /* ---- make run.be executable after copying ----
                    system ("chmod ug+x run.be");              */

        sprintf (pch, "chmod ug+x run.be");
printf ("EXECUTING: %s\n", pch);
        system (pch);
```

```
        for (i = 0; i < MAX_BACK_ENDS; i++)
            {
            /* ---- create new version directory on each BackEnd ---- */
            /* ------------------  could test before execution  -----
               rsh %s mkdir current_version                          */

            sprintf(pch,"rsh %s -n mkdir be.%s", be_names[i], current_version);
printf ("EXECUTING: %s\n", pch);
            system (pch);

            /* ---- copy run.be to new directory on each BackEnd ---- */
            /* rcp run.be %s/u/mdbs/be.%s */
            sprintf (pch, "rcp run.be %s:%s ", be_names[i], BE_path);

printf ("EXECUTING: %s\n", pch);
            system (pch);

            /* ---- copy 6 processes to new directory on each BackEnd ---- */
            /* rcp ../BE/*.exe %s:/u/mdbs/be.%s */
            sprintf (pch, "rcp /u/mdbs/%s/BE/*.exe %s:%s ",
                        current_version, be_names[i], BE_path);

printf ("EXECUTING: %s\n", pch);
            system (pch);

        }
     }
     else
         /* user did not wish to change current version */
         return (NO);

    return (YES);

}   /* end change_version () */
```

```
/* ------------------------------------------------------------------
 * RECONFIGURE allows user to change configuration.
 *               no illegal configurations are accepted.
 *               assumes get_configuration completed successfully
 *
 *  parameters:  pN_BackEnds      (in)   the number in current Configuration
 *               current_BackEnds[]      strings of each back end name
 *
 *  returns:     YES  if new version
 *               NO   if no change to version
 */
int reconfigure (pN_BackEnds, current_BackEnds)
int *pN_BackEnds;
char *current_BackEnds[];
{
    char pch[MAX_STRING];              /* temporary pointer to string     */
    FILE *pFILE;                       /* pointer to .config.db file      */
    int reconfiguring = TRUE;          /* boolean, if BE is a valid one   */
    int i;                             /* array index                     */

    /* NOTE:  if we want to re arrange data on back-ends, we must save
              the old current_BackEnds[1..N_BackEnds] data to a tape
              then reload (via tape or file--mass load) so mdbs will
              automatically redistribute.  NOT IMPLEMENTED               */
```

```
    /*      ----------  get new configuration                                */
    while (reconfiguring)
    {

        printf (" Indicate which Back Ends you want in the configuration \n");
        printf (" by responding y/n as each Back End is listed:\n");

        for (*pN_BackEnds = 0, i=0; i< MAX_BACK_ENDS; i++)
        {
            printf ("\n\t\t%s ? ", be_names[i]);
            scanf  ("%s", pch);
            if (*pch == 'y' || *pch == 'Y')
            {
                strcpy (current_BackEnds[*pN_BackEnds], be_names[i]);
                (*pN_BackEnds)++;
            }
        }
        show_configuration (*pN_BackEnds, current_BackEnds, current_version);

        /* -------- opportunity to confirm changes -------- */
        if (*pN_BackEnds == 0)
            {
            printf ("WARNING: ");
            printf ("If you do not specify any back ends, the configuration\n");
            printf ("\t  will not be updated\n\n");
            }

        printf ("Is this configuration satisfactory? (y/n) ");
        scanf  ("%s", pch);
        printf ("\n");
        if (*pch == 'y' || *pch == 'Y')
        {
            reconfiguring = FALSE;
        }
    }   /* end while (reconfiguring) */

    /* ----------  write new .config.db only if valid -------- */

    if (*pN_BackEnds == 0)
        /* reload old configuration since new one is invalid */
        get_configuration (pN_BackEnds, current_BackEnds, current_version);
    else
        if (pFILE = fopen (".config.db", "w"))
        {
            fprintf (pFILE, "%s\n", current_version);

            for (i=0;  i < *pN_BackEnds; i++)
            {
                fprintf (pFILE, "%s\n", current_BackEnds [i]);
            }
            fclose (pFILE);
        }
        else
        {
            printf ("Error:  unable to write to .config.db\n");
            printf ("\t.config.db not updated with change(s)\n\n");
            printf ("\tYou must do one of the following:\n");
            printf ("\t\tchmod u+w .config.db  \n\t\trm .config.db\n");
            printf ("\tthen run this program again.\n");
            return (ERROR);
        }
    /* ----------  display configuration                                  */
    /* ------------------------------------------------------             */
    show_configuration (*pN_BackEnds, current_BackEnds, current_version);

    zero_backends (*pN_BackEnds, current_BackEnds);
    return (NO_ERROR);
}   /* end reconfigure ()  */
```

```
/* ------------------------------------------------------------------
 * MAIN        parameters:  command line arguments (count and strings)
 *
 *               see comments at beginning of file, and in code
 */
main (argc, argv, env)
int argc;
char *argv[];
char *env[];
{
    int  n_BackEnds = 0;     /* number of Back Ends being used          */
    char pch[MAX_STRING];    /* temporary pointer to string             */
    char pwd[MAX_STRING];    /* current directory, pointer to string    */
    int  i;                  /* array index                             */
    int new_ver    = FALSE; /* if -v option used to call this program */
    int new_config = FALSE; /* if -c option used to call this program */
    int dist_code  = FALSE; /* if -d option used to call this program */

    char current_version[MAX_STRING];  /* read in from .config.db file        */

    char BE_path[MAX_STRING];     /* see set_path (); "/u/mdbs/be.VERSION/";    */
    char CNTRL_path[MAX_STRING]; /* see set_path (); "/u/mdbs/VERSION/CNTRL/";*/

    printf ("\n\nWelcome to MDBS, today is  ");
    fflush (stdout);
    system ("date");

    printf ("\n\n");

    /* ----------  check for command line parameters                   */
    if (check_for_flags (argc, argv, &new_ver, &new_config, &dist_code ))
        return (ERROR);

    /* ----------  read .config.db                                     */

    if (get_configuration (&n_BackEnds, current_BackEnds, current_version))
    {
        printf ("\nError in get_configuration, Exiting configure program.\n");
        return (ERROR);
    }

    getwd  (pwd);

    if (not_in_pwd (pwd, current_version))
    {
        printf ("\nWARNING:  version name \"%s\" from .config.db", current_version);
        printf ("\n\t  not in current path: %s\n", pwd);
        printf ("\t  MDBS will not run unless version name is in pwd\n");
    }

    set_paths (current_version, BE_path, CNTRL_path);

    /* ----------  display executable listings of BE and CNTRL         */
    printf ("\nCheck the time each file was last compiled:\n\n");

    sprintf (pch, "ls -l ../BE/*.exe");
    system  (pch);

    printf ("\n");

    sprintf (pch, "ls -l ../CNTRL/*.exe");
    system  (pch);

    printf ("\nThere should be 12 files listed, if not you need to recompile.\n");
```

```
    /* ----------  ask to compile  if -d and -v not used to call main */
    /* ------------------------------------------------------------- */
    if (dist_code  == FALSE && new_ver == FALSE) {
        printf ("\nDo you need to recompile any executable and/or copy the 6 ");
        printf ("\n executable files to each Back End ");
        printf ("\n(bget, bput, cc, dio, dirman, recp.exe)? (y/n) ");

        scanf ("%s", pch);
        printf ("\n");
    }
    if (*pch == 'y' || *pch == 'Y' || dist_code )
    {
        printf ("\nDo you need to recompile before copying to the back ends? (y/n) ");

        scanf  ("%s", pch);
        printf ("\n");
        if (*pch == 'y' || *pch == 'Y')
        {
            /* -------- recompile executable code                        */
            /* ---------------------------------------------------------- */
            sprintf (pch, "../bin/make all");
printf ("EXECUTING: %s\n", pch);
            system  (pch);

        }
        /* -------- copy recompiled code to each back end               */
        /* ------------------------------------------------------      */
        sprintf (pch, " ");
        for (i = 0; i < MAX_BACK_ENDS; i++)
            {
            /* for each back end being used, stop 6 processes on it
               which may still be running from last time               */

             sprintf (pch, "rcp ../BE/*.exe %s:%s", be_names[i], BE_path);
printf ("EXECUTING: %s\n", pch);
             system  (pch);

            }
    }       /* end if recompile or recopy BE processes                 */


    /* ------ to change the current version  -v option must be used    */
    /* ------------------------------------------------------------------ */
    if (new_ver)
    {
        change_version (&n_BackEnds,     current_BackEnds,
                        current_version, BE_path, CNTRL_path, pwd);
    }

    show_configuration (n_BackEnds, current_BackEnds, current_version);


    /* ----------  ask to reconfigure if -c not used                */
    /* ------------------------------------------------------------ */
    if (new_config == FALSE) {
        printf ("\n WARNING:  All data will be lost if you reconfigure\n");
        printf ("\n Do you wish to reconfigure the Back Ends? (y/n) ");
        scanf  ("%s", pch);
        printf ("\n");
    }

    if (*pch == 'y' || *pch == 'Y' || new_config)
    {
        if (reconfigure (&n_BackEnds, current_BackEnds) == ERROR)
            return (ERROR);
    }
```

```
else  /*  -------- do not want to reconfigure back ends -----  */
{
    /* ----------  ask to use present database --------------  */
    /* ------------------------------------------------------  */
    /* ----------  display currently loaded database (pry)     */
    /* ------------------------------------------------------  */
    printf ("\n Do you wish to use current database? (y/n) ");
    scanf  ("%s", pch);
    printf ("\n");

    if (*pch == 'y' || *pch == 'Y')
    {
        /* assumes that the user knows what database is loaded    */
        /* ------------------------------------------------------ */
        /* could check for existing data on disks here

        printf ("Checking what database is on Back Ends is ");
        printf ("NOT IMPLEMENTED\nYou have to remember or use pry\n");*/
        ;
    }
    else
    {
        zero_backends (n_BackEnds, current_BackEnds);
    }

}  /* end else of   if (reconfigure?)                          */


printf ("\n Do you wish to run the Multi Modal, Multi Lingual,\n");
printf ("\t\tMulti Backended Database System? (y/n) ");
scanf  ("%s", pch);
printf ("\n");


if (*pch == 'n' || *pch == 'N')
{
    return (NO_ERROR);
}


/* --------- verify current_version in pwd ---------------------  */
/* ---------  explain what to do and ABORT if it is not          */
if (not_in_pwd (pwd, current_version))
{
    printf ("\nERROR:  version name \"%s\" from .config.db", current_version);
    printf ("\n\tis not in current path: %s\n\n", pwd);

    printf ("\tIf version name is not correct, restart with -v option\n");
    printf ("\tto correct version name.\n\n");
    printf ("\tIf version name is correct, then you are running main\n");
    printf ("\tfrom the wrong directory.  You must either\n");
    printf ("\t\trun main from /mdbs/%s/run    or\n", current_version);
    printf ("\t\tchange begin / mdbs / start alias to cd to /mdbs/");
    printf ("%s/run\n\n", current_version);

    printf ("\tWhen changing versions, remember to verify default_version");
    printf (" is \n\tcorrect in configure.h.  If a change is made,\n");
    printf ("\trecompile main.c then copy main to /mdbs/");
    printf ("%s/run/main\n\n", current_version);
    printf ("\tExiting configure program.\n");
    return (ERROR);
}
```

```
    /* --------- verify current_version matches default_version  ---  */
    /* ---------  explain what to do and ABORT if no match   -------  */
    if (strncmp (default_version, current_version, strlen (current_version)))
    {
        printf ("\nERROR:");
        printf ("\tdefault_version \"%s\" does not match \n", default_version);
        printf ("\tcurrent_version \"%s\"\n", current_version);
        printf ("\tIf current_version is correct,\n");
        printf ("\t\tGo to /mdbs/%s/bin directory\n", current_version);
        printf ("\t\tIn configure.h correct default_version \n");
        printf ("\t\tcompile main.c by typing 'make main'\n");
        printf ("\t\tmakefile will automatically copy main to run/main\n\n");
        printf ("\tIf current_version is not correct, restart with -v option\n");
        printf ("\tto correct version name.\n\n");
        printf ("\tExiting configure program.\n");
        return (ERROR);
    }

    /* ------------ kill all lingering processes ------------------
       ------------------------------------------------------------ */

    for (i = 0; i < n_BackEnds; i++)
        {
        /* for each back end being used, stop 6 processes on it
           which may be still running from last time              */

            sprintf (pch, "stop.%s >>$! trace", current_BackEnds[i]);
            system  (pch);
        }

    /* ensure the 6 processes are stopped on the controller       */

    sprintf (pch, "stop.%s >>$! trace", controller);
    system  (pch);

    /* ----------  start 5 controller processes ----------------
       ------------------------------------------------------------ */
printf ("EXECUTING: start.cntrl\n", pch);
        system ("start.cntrl");

    /* NOT IMPLEMENTED:   check all 5 processes running           */

    /* ----------  start all back end processes on each back end  */
    /*             execute:  "rsh %s -n /u/mdbs/be.greg/run.be"   */
    /* ------------------------------------------------------     */

    for (i = 0; i < n_BackEnds; i++)
    {
        sprintf (pch, "rsh %s -n %srun.be &", current_BackEnds[i], BE_path);
printf ("EXECUTING: %s\n", pch);
        system (pch);

    }

    /* NOT IMPLEMENTED:  check all processes running on each back end */

    /* ------  start user interface on (this machine, the) controller */
    /*         execute:  ti.exe 'n_BackEnds'                          */
    /* ------------------------------------------------------         */

    sprintf (pch, "%s%s %d", CNTRL_path, CNTRL_process [0], n_BackEnds);

printf ("EXECUTING: %s\n", pch);
    system (pch);

    return (NO_ERROR);
}   /* end main() */
/* eof:  main.c */
```

APPENDIX F. MISCELLANEOUS FACTS ABOUT USING MDBS

A substantial portion of this appendix was based on the README file written by John Locke, systems programmer at the Naval Postgraduate School.

A. HOW TO START-UP THE SYSTEM AFTER THE POWER GOES OFF.

1. For db1 - db9

Turn on monitors (for all systems)
Turn on db8 (use switch above tape drive)

wait about 10 seconds
push left of 4 red horizontal buttons (1' from floor)

At monitor type
@ <CR> (<CR> means carriage return)

you should not get prompted for the date since an internal battery exists

For each Back end Unit:
Turn on power switch centered on the front top of the machine)
wait
Push START button in vertical column of brown buttons)

wait for green light
Push left of 4 red horizontal buttons (1' from floor)

At monitor type
@ <CR>

when prompted for date (you only get 30 seconds, or the shut-down time becomes the current time)
enter date in format: YYMMDDHHMM

These instructions are short (and cryptic) since most people do not use them, and the typical person does not want to read long instructions.

2. For Sun4 Workstations

Turn on power. If automatic reboot does not start, hold down the STOP (or F1) button then push the A button. At the '>' prompt, enter 'b' for boot. If the system does not reboot, then seek help from a systems programmer.

B. SOURCE DIRECTORY LOCATION

This source distribution is in directory /u/mdbs/greg on host db8. The greg version is an improvement on source from directory /u/mdbs/7 on host db4. Version 7 is protected by having root as the owner. To work on the system it is STRONGLY recommended to make a copy of the whole distribution or at least the file or files which you modify BEFORE making any changes. For example, to copy the source into a directory named "new" execute this command (which will work across the network):

```
> rcp -r db8:/u/mdbs/greg new
```

The rest of these instructions will assume the "new" is the new source. You will want to use a more distinctive name, like your own. Expect the copying to take about 3 minutes.

C. COMPILING

To compile or recompile all or part of MDBS, change directory to the bin directory of the version you need compiled. Then type "**make all**" to compile any code which might have been modified[1]. This includes any utility programs stored in any subdirectory which has corresponding lines in a makefile to compile them. Other options for the main makefile include auto, clean, and dist.

make auto will compile only the system-generation program (aka configure.h and main.c).

make clean will remove all .o files

make dist will compress all files into /u/mdbs/dist.tar and store that file to tape

When you type **make all**, a file with shell commands, makeall, is executed. This file calls the mk file in directories corresponding to each of the 12 processes. Ensure all

1. It is possible that a modified file will not cause the makefile to recompile the associated process. This may be caused by a missing or incorrect dependency in the corresponding makefile. You should check the make file and correct the dependency (or force recompililation by deleting the object file for the affected file and some or all of the object and executable files compiled from that file). Then make the affected file.

controller processes are not running by using the automatic system start up program or by using the stop.db8 file to kill them.

A successful compilation should look something like:

```
> cd bin
> make all
Making the Configure Program
Making IIG...
Making CPCL...
Making PP...
Making REQP...
Making TI...
Making CC...
Making DIO...
Making DM...
Making BPCL...
Making RECP...
Controller Processes...
-rwxrwxr-x  1 mdbs        75398 May 21 15:18 ../CNTRL/cget
-rwxrwxr-x  1 mdbs        75398 May 21 15:18 ../CNTRL/cput
-rwxrwxr-x  1 mdbs        56657 May 21 15:17 ../CNTRL/iig
-rwxrwxr-x  1 mdbs        56947 May 21 15:20 ../CNTRL/pp
-rwxrwxr-x  1 mdbs        92551 May 21 15:23 ../CNTRL/reqp
-rwxrwxr-x  1 mdbs       116578 May 21 15:26 ../CNTRL/ti
Backend Processes...
-rwxrwxr-x  1 mdbs        75414 May 21 15:36 ../BE/bget
-rwxrwxr-x  1 mdbs        75414 May 21 15:36 ../BE/bput
-rwxrwxr-x  1 mdbs        67607 May 21 15:29 ../BE/cc
-rwxrwxr-x  1 mdbs        46655 May 21 15:30 ../BE/dio
-rwxrwxr-x  1 mdbs        93213 May 21 15:34 ../BE/dirman
-rwxrwxr-x  1 mdbs       108419 May 21 15:39 ../BE/recp
```

The output from the several make files are put in either err (on cs4322 VerE.6) or make_result (on mdbs version 7 and later) files corresponding to the mk file. The make_result files can be viewed by typing:

```
> cd /u/mdbs/new
> more */*/make_result
```

REMEMBER to copy the back end processes to the back ends each time you recompile since the back end executables are stored on the controller and are not copied to each back end automatically. The automatic system startup program will do the copying for you (see Appendix B, paragraph C). This copying should be done each recompilation since any modification to an include file could affect inter-process communications. A successful make will change the date and time of the processes whose code was modified.

D. COMPILING AN INDIVIDUAL PROCESS

To compile an individual process, change directory to the source for that process and execute the script "mk". "Mk" will change into the "Object" directory, run the makefile and, if successful, will compile the updated process and place it in the CNTRL or BE directory as appropriate. The results of the make will be written to the file "**make_result**" in the process source directory. It *should always be read after compiling to see what actually happened.* In the event of an unsuccessful compile, it will contain the error message.

E. DEBUGGING AND FLAGS

Most source files have a line near the top to include the file"flags.def":

```
#include "flags.def"
```

This causes the preprocessor to replace the #include line with the contents of "flags.def". An abbreviated version of "flags.def":

```
/*
#define EnExFlag
#define msg_pr_flag
*/
```

Normally, the #define lines in "flags.def" will be bracketed with the C comment delimiters. There is one important exception, #define LangIF_Flag, which must not be commented out. If LangIF_Flag is not defined, then you will be unable to use any language other than the attribute based data language. A comment has the effect of turning off the flag(s). To turn one on, it is necessary to edit "flags.def" and move the selected flags outside of the comment delimiters. This will not force a recompile of the entire process because "flags.def" is not listed as a dependency for the source files in the makefile. We don't always want every process compiled with flags turned on. It can create a great deal of debugging output, and we may have isolated the problem area of the code by then. So after editing "flags.def", the next step is to remove the object files for the modules we want flags turned on for. These files, ending with .o, match the names of the .c files and are to be found

in the "Object" directory below the process source. After removing the relevant .o files, build the process with "mk".

So what does turning on a flag do? Say I edited "flags.def" to be:

```
#define EnExFlag
/*
#define msg_pr_flag
*/
```

The preprocessor variable EnExFlag is now defined to a logical true. The preprocessor will act conditionally upon encountering structures like this:

```
#ifdef EnExFlag
   printf("Enter func_x\n");
   fflush(stdout);
#endif
```

If EnExFlag is true, the code between the #ifdef and #endif lines will be included in the compiled object. The EnExFlag, it so happens, is used to show when the entrance and exit to functions occurs in the flow of execution. Other flags have different purposes--these are documented in "flags.def".

To turn off flags requires a simple reversal of the procedure. Place the #define lines back between the comment delimiters in "flags.def", remove the object files that have flags turned on, and recompile the process.

REMEMBER that there are multiple copies of some include files, particularly flags.def. If you make a change to one, you may or may not want to change the other copies of the files depending on what stage of programming or debugging you are in.

F. THE HIGHER LEVEL LANGUAGE INTERFACES

In version 7, the make files are set up to compile the test interface (TI) without the four higher level language interfaces, since not everyone works with them and to debug the controller-back end communications. To include this code these two steps must be followed:

1. Edit flags.def in TI to define the LangIF_Flag. See the above section on debugging flags.

2. Edit TI/makefile and uncomment the LIBS definition and the line with "cd ../LangIF". Both are marked with comments.

3. "Make all" in Version/bin as described above.

To turn off the higher level language interfaces. reverse the above steps in the same order listed.

G. CHECKING THE HARDWARE CONFIGURATION

The system is typically run on a controller (usually db8) and at least one back end. The controller requires no special hardware, but is named in the file COMMON/pcl.def. Change this file to use a controller other than db8. Unlike db8, each back end must have two disk drives to be used as the meta and data disks, /dev/sd1c and /dev/sm0c, respectively. Not all the ISI's have both disks in working order. To determine if the disks are available, perform a simple read operation on each, for example:

```
> head /dev/sd1c
```

The prompt should return quickly. If the following message prints, the disk is not available and the machine cannot be used as a back end:

```
> head /dev/sm0c
/dev/sm0c: No such device or address
```

If this message appears, the ownership of the disk file must be changed by a staff member:

```
> head /dev/sd1c
/dev/sd1c: Permission denied
```

at the time of this writing, the status of equipment is:

db1 Pt. Mugu
db2 SUPSHIP San Francisco
db3 back end, working
db4 back end, working
db5 no meta disk
db6 no data disk
db7 back end, working
db8 controller, working
db9 no data disk

H. THE DEVELOPMENT CYCLE

This is to give the big picture of working on the MDBS code. There are a few steps to the development cycle. Apart from the actual coding, all are easy but it is easy to forget a step and then be confused by the results of a run. The following will provide a checklist of steps to follow. It assumes that the initial set-up has already been done and the system has been successfully run.

1. Edit code
2. Compile process with individual process"mk" or entire system "make all"
3. Check trace files for debugging output
4. If any include file or back end code was modifed, copy the new back end process(es) to the directory on each back end corresponding to the version name (e.g. be.greg). This can be done by the system-generation software (Appendix B, section C).
5. Run the system (as described in Appendix B, section A).

I. RUNNING THE SYSTEM

To start the system, see Appendix B. Once the user-interface process is running, the system might appear to be up, though some processes have died. This will usually be due to changes you have made to the code not working. For example, to check the process table on a back end:

```
> ps ax | grep ".exe"
 3956 ?  S      0:00 /u/mdbs/be.Version_name/bget.exe
 3957 ?  S      0:00 /u/mdbs/be.Version_name/dirman.exe 100
 3958 ?  S      0:00 /u/mdbs/be.Version_name/cc.exe
 3959 ?  S      0:00 /u/mdbs/be.Version_name/recp.exe
 3960 ?  I      0:00 /u/mdbs/be.Version_name/dio.exe
 3961 ?  S      0:00 /u/mdbs/be.Version_name/bput.exe
```

The output shows the six back end processes running. The run scripts are set to channel their output, if any, to files with the name "proc_name.tr", where proc_name is the name of the process. If a process is compiled to produce debugging data, the data will be output to that file. (ti.exe, the "test" user interface, is an exception--it outputs to the screen. To save TI output, execute "script" before starting the program.)

Note that the distribution assumes use of only one back end. The number of back ends is given to ti as an argument when called by the system-generation program.

J. UTILITY PROGRAMS

Several utility programs have been written for mdbs. The executable file for each is usually kept in the bin directory of the version. Location of source is noted for binaries.

constants - prints out meta table sizes (source in DM)
cpcount - copies a file for a given number of bytes; used to copy meta and data disks to other back ends (source in DIO)
cpydisks - shell script that uses cpcount; WARNING: Do not run this program from the controller
disp - dynamic meta table disp (source in DM)
gdb - generate test database of arbitrary size from template file (source in CNTRL/TI/). usage: gdb D.t D[.r]
makeall - script to make every process (called by bin/makefile)
rectag - prints out formatted data from data disk (source in DIO)
zero - writes all 0's to a file; used to blank meta and data disks (source in DIO)

APPENDIX G. SELECTED MDBS FILE LISTINGS

A. CONTROLLER

The following files are kept in the 'run' directory for each version.

```
# file:  run/master.run.be

#!/bin/csh

echo Running backend on 'hostname'...

/u/mdbs/be.VERSION/bget.exe              >&! /u/mdbs/be.VERSION/bget.tr &
/u/mdbs/be.VERSION/dirman.exe 100        >&! /u/mdbs/be.VERSION/dirman.tr&
/u/mdbs/be.VERSION/cc.exe                >&! /u/mdbs/be.VERSION/cc.tr &
/u/mdbs/be.VERSION/recp.exe              >&! /u/mdbs/be.VERSION/recp.tr &
/u/mdbs/be.VERSION/dio.exe               >&! /u/mdbs/be.VERSION/dio.tr &

(sleep 16; /u/mdbs/be.VERSION/bput.exe >&! /u/mdbs/be.VERSION/bput.tr)&
```

master.run.be

```
# file:  run/stop.db4

# echo stopping processes on isiv4
rsh db4 /u/mdbs/bin/stop.cmd
```

stop.db4

```
# file:  run/stop.db8

echo stopping processes on 'hostname', the controller

ps -x | awk -f .exe.awk >! list2.stop
# file .exe.awk contains:  /'exe/ {print $1}

set noclob

set a='cat list2.stop'
if ($#a) then
    echo killing $a
    kill $a
else
    echo no processes to kill on 'hostname'
endif
rm list2.stop
```

stop.db8

```
# file:  run/zero.db8

# this is only for the controller

#!/bin/csh

echo Removing CINBT and IIG AT tables on controller, 'hostname'...
rm -f /u/mdbs/UserFiles/*.cinbt
rm -f /u/mdbs/UserFiles/*.iig.at
rm -f /u/mdbs/UserFiles/.R*

rm -f /u/mdbs/.*.pid
rm -f /u/mdbs/greg/run/trace/*.tr

# removing ? from Sockets:  copied from cs4322 acct
rm -f /u/mdbs/Sockets/G_PCLB
rm -f /u/mdbs/Sockets/G_PCLC

# there is no backend meta disk on db8

# there is no backend disk on db8
```

zero.db8

```
# file:    /greg/run/start.cntrl.screen.trace

# THIS FILE WRITES ALL DEBUGGING STATEMENTS TO THE SCREEN

#!/bin/csh

echo starting 5 of 6 controller processes on 'hostname'...

../CNTRL/cget.exe &
../CNTRL/cput.exe &
../CNTRL/pp.exe   &
../CNTRL/iig.exe  &
../CNTRL/reqp.exe &

# to be executed when called by program which called this shell script
# output goes to screen
# ../CNTRL/ti.exe n
```

start.cntrl

```
# file:    /greg/run/start.cntrl

# NORMAL METHOD is to put debugging comments into .tr files

#!/bin/csh

echo starting 5 of 6 controller processes on 'hostname'...

../CNTRL/cget.exe       >&! trace/cget.tr &
../CNTRL/cput.exe       >&! trace/cput.tr &
../CNTRL/pp.exe         >&! trace/pp.tr &
../CNTRL/iig.exe        >&! trace/iig.tr &
../CNTRL/reqp.exe       >&! trace/reqp.tr &

# to be executed when called by program which called this shell script
# output goes to screen
# ../CNTRL/ti.exe n
```

start.cntrl

Two files modified for Sun csh

```
# file:  run/stop.db11
# modified 930427 for compatability with Sun system.

echo stopping processes on 'hostname', the controller

# !/bin/csh

ps -x | awk -f .exe.awk > list2.stop
# file .exe.awk contains:  /\.exe/ {print $1}

set noclob

foreach a ('cat list2.stop')
echo killing $a
     kill $a
end
rm list2.stop
```

stop.db11

```
# file:    /greg/run/start.cntrl
# modified 930512 for compatability with Sun.

#!/bin/csh
rm trace/*.tr
echo starting 5 of 6 controller processes on 'hostname'...

../CNTRL/cget.exe       > trace/cget.tr &
../CNTRL/cput.exe       > trace/cput.tr &
../CNTRL/pp.exe         > trace/pp.tr &
../CNTRL/iig.exe        > trace/iig.tr &
../CNTRL/reqp.exe       > trace/reqp.tr &

# to be executed when called by program which called this shell script
# output goes to screen
# ../CNTRL/ti.exe n
```

start.cntrl

B. BACK ENDS

The following file is stored on each back end in the 'bin' directory.

```
# file: binstop.cmd

echo stopping processes on back end 'hostname'

ps -x | awk -f bin/exe.awk >! list2.stop
# file exe.awk contains:   /'exe/ {print $1}

set noclob

set a='cat list2.stop'
if ($#a != 0) then
    echo killing $a
    kill $a
    #  for some reason, this won't work on last line  >$! stop.trace
else
    echo no processes to kill on 'hostname'
endif
rm list2.stop
```

stop.cmd

The following file is stored on each back end in the 'be.VERSION' directory.

```
#!/bin/csh

echo Running backend on 'hostname'...

/u/mdbs/be.greg/bget.exe                >&! /u/mdbs/be.greg/bget.tr &
/u/mdbs/be.greg/dirman.exe 100          >&! /u/mdbs/be.greg/dirman.tr&
/u/mdbs/be.greg/cc.exe                  >&! /u/mdbs/be.greg/cc.tr &
/u/mdbs/be.greg/recp.exe                >&! /u/mdbs/be.greg/recp.tr &
/u/mdbs/be.greg/dio.exe                 >&! /u/mdbs/be.greg/dio.tr &

(sleep 16; /u/mdbs/be.greg/bput.exe     >&! /u/mdbs/be.greg/bput.tr) &
```

run.be

LIST OF REFERENCES

Banerjee, Jayanta, David K. Hsiao, and Krishnamurthi Kannan, "DBC--A Database Computer for Very Large Databases," IEEE Transactions on Computers, vol C-28, no 6, June 1979, pp. 414-429.

Bourgeois, Paul A., "The Instrumentation of the Multimodel and Multilingual User Interface," M. S. Thesis, Naval Postgraduate School, Monterey, California, March, 1993.

Chung, Chin-Wan, "DATAPLEX: An Access to Heterogeneous Distributed Databases," Communications of the ACM, vol 33, no 1, January 1990, pp. 70-80.

De Witt, David J. "DIRECT--A Multiprocessor Organization for Supporting Relational Database Management Systems," IEEE Transactions on Computer, vol C-28, No. 6, June 1979, pp 395-406.

Elmasri, Ramez and Shamkant B. Navathe. Fundamentals of Database Systems. New York, The Benjamin/ Cummings Publishing Company, Inc., 1989.

Hammond, Greg Alan, "The Instrumentation of a Parallel, Distributed Database Operation, Retrieve-Common, for Merging two large sets of records," M. S. Thesis, Naval Postgraduate School, Monterey, California, June 1992. (H17526)

Hennessy, John L. and David A Patterson. Computer Architecture A Quantitative Approach. San Mateo, California. Morgan Kaufmann Publishers, Inc., 1990.

Hsiao, David K., "A Parallel, Scalable, Microprocessor-Based Database Computer for Performance Gains and Capacity Growth," IEEE Micro, December 1991, pp. 44-60.

Hsiao, David K., "Federated Databases and Systems: Part I - A Tutorial on Their Data Sharing," Very Large Database (VLDB) Journal, vol 1, no 1, 1992, pp. 127-179.

Hsiao, David K., "Federated Databases and Systems: Part II - A Tutorial on Their Resource Consolidation," Very Large Database (VLDB) Journal, vol 1, no 2, 1992, pp. 285-310.

.Hsiao, David K. and Magdi N. Kamel, "Heterogeneous Databases: Proliferations, Issues, and Solutions," IEEE Transactions of Knowledge and Data Engineering, vol 1, no 1, March 1989, pp. 45-62.

Quantock, David E., "The Real-Time Roll-Back and Recovery of Transactions in Databae Systems," M. S. Thesis, Naval Postgraduate School, Monterey, California, June, 1989.

Schuster, Stewart A. et.al. "RAP.2--An Associative Processor for Databases and Its Applications," IEEE Transactions on Computers, Vol C-28, No 6, June 1979, pp. 446-458.

Sheehan, William A., "The Design of a DL/I-to-Network Interface for the Multi-Model, Multi-Lingual, Multi-Backend Database System," M. S. Thesis, Naval Postgraduate School, Monterey, California, December 1989.

Su, Stanley Y. W., Le Huu Nguyen, Ahmed Emam, and G. J. Lipovski, "The Architectural Features and Implementation Techniques of the Multicell CASSM," IEEE Transactions on Computers, vol C-28, no. 6, June 1979, pp. 430-445.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
Dudley Knox Library Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
Chairman, Code CS Computer Science Department Naval Postgraduate School Monterey, CA 93943	2
Professor David K. Hsiao Computer Science Department Naval Postgraduate School Monterey, California 93943	2
Ms. Doris Mlezco Code 9033 Naval Pacific Missile Test Center Point Mugu, CA 93042	2
Lieutenant Andrew P. Meeks, USN 105 Mikel Court Summerville, SC 29483	1